

CONFERENCES IN RESEARCH AND PRACTICE IN
INFORMATION TECHNOLOGY

VOLUME 62

COMPUTER SCIENCE 2007

AUSTRALIAN COMPUTER SCIENCE COMMUNICATIONS, VOLUME 29, NUMBER 1.



AUSTRALIAN
COMPUTER
SOCIETY



CO_{mputing}
R_{esearch}
& E_{ducation}

COMPUTER SCIENCE 2007

Proceedings of the
Thirtieth Australasian Computer Science Conference
(ACSC2007),
Ballarat, Victoria, Australia,
January 30 to February 2, 2007

Gillian Dobbie, Ed.

Volume 62 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.

Proceedings of the Thirtieth Australasian Computer Science Conference (ACSC2007), Ballarat, Victoria, January 30 to February 2, 2007

Conferences in Research and Practice in Information Technology, Volume 62.

Copyright ©2007, Australian Computer Society. Reproduction for academic, not-for profit purposes permitted provided the copyright text at the foot of the first page of each paper is included.

Editors:

Gillian Dobbie

Department of Computer Science
The University of Auckland
Private Bag 92019
Auckland,
New Zealand
Email: gill@cs.auckland.ac.nz

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 62.
ISSN 1445-1336.
ISBN 1-920-68243-0.

Printed, December 2006 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 Thirtieth Australasian Computer Science Conference (ACSC2007), Ballarat, Victoria, January 30 to February 2, 2007

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 ACSC 2007 Sponsors	xv

Keynote

Just Like Magic: Anthropological Musings on the History and Culture of Wireless Technologies	3
<i>Genevieve Bell</i>	

Contributed Papers

Algorithms

Compact Layout of Layered Trees	7
<i>Kim Marriott and Peter Sbarski</i>	
Improved Shortest Path Algorithms For Nearly Acyclic Directed Graphs	15
<i>Lin Tian and Tadao Takaoka</i>	
From Graphs to Euclidean Virtual Worlds: Visualization of 3D Electronic Institutions	25
<i>Sara Drago, Anton Bogdanovych, Massimo Ancona, Simeon Simoff and Carles Sierra</i>	

Communications and Networks

QUIP: A Protocol For Securing Content in Peer-To-Peer Publish/Subscribe Overlay Networks	35
<i>Amy Corman, Peter Schachte and Vanessa Teague</i>	
Enhancing Data Locality in a Fully Decentralised P2P Cycle Stealing Framework	41
<i>Richard Mason and Wayne Kelly</i>	
Hybrid Mesh Ad-Hoc On-Demand Distance Vector Routing Protocol	49
<i>Asad Pirzada, Marius Portmann and Jadwiga Indulska</i>	

Databases

Searching With Style: Authorship Attribution in Classic Literature	59
<i>Ying Zhao and Justin Zobel</i>	
On Inferences of Full Hierarchical Dependencies	69
<i>Sven Hartmann and Sebastian Link</i>	

Domination Normal Form - Decomposing Relational Database Schemas	79
<i>Henning Koehler</i>	
Software Engineering	
Jooj: Real-Time Support For Avoiding Cyclic Dependencies	87
<i>Hayden Melton and Ewan Tempero</i>	
HAT-Trie: A Cache-Conscious Trie-Based Data Structure For Strings	97
<i>Nikolas Askitis and Ranjan Sinha</i>	
Web Services Discovery Based On Schema Matching	107
<i>Yanan Hao and Yanchun Zhang</i>	
Mobile Computing and Security	
Conflict Management For Real-Time Collaborative Editing in Mobile Replicated Architectures	115
<i>Sandy Citro, Jim McGovern and Caspar Ryan</i>	
Architecture of a Web Accelerator For Wireless Networks	125
<i>Jian Song and Yanchun Zhang</i>	
Periodical Payment Model Using Restricted Proxy Certificates	131
<i>Grigori Goldman</i>	
Graphics and Image Processing	
Mutually Visible Agents in a Discrete Environment	141
<i>Joel Fenwick and Vladimir Estivill-Castro</i>	
Exploring Human Judgement of Digital Imagery	151
<i>Timo Volkmer, James A. Thom and Seyed M.M. Tahaghoghi</i>	
Segmentation and Border Identification of Cells in Images of Peripheral Blood Smear Slides	161
<i>Nicola Ritter and James Cooper</i>	
Security	
Cross-Layer Verification of Type Flaw Attacks on Security Protocols	171
<i>Benjamin Long, Colin Fidge and David Carrington</i>	
Access Control Models and Security Labelling	181
<i>Chuchang Liu, Angela Billard, Maris Ozols and Nikifor Jeremic</i>	
Cost-Based Framework and Simulation of DoS-Resistant Protocols Using Coloured Petri Nets	191
<i>Suratose Tritilanunt, Colin Boyd, Ernest Foo and Juan Manuel González Nieto</i>	
Software Engineering and Simulation	
The CRSS Metric for Package Design Quality	201
<i>Hayden Melton and Ewan Tempero</i>	
Dynamic Measurement of Polymorphism	211
<i>Kelvin (Hio Tong) Choi and Ewan Tempero</i>	
Efficient Cycle-Accurate Simulation of the Ultrasparc III CPU	221
<i>Peter Strazdins, Bill Clarke and Andrew Over</i>	
Author Index	229

Preface

The Australasian Computer Science Conference (ACSC) series is an annual forum, bringing together research sub-disciplines in Computer Science. The meeting allows academics and researchers to discuss research topics as well as progress in the field, and policies to stimulate its growth. This volume contains papers presented at the Thirtieth ACSC in Ballarat, Victoria, Australia. ACSC 2007 is part of the Australasian Computer Science Week which ran from Jan 30th to Feb 2nd, 2007.

The ACSC 2007 call for papers solicited contributions in all areas of computer science research. This year's conference received submissions from Australia, New Zealand, China, Japan, and Taiwan. The topics addressed by the submitted papers illustrate the broadness of the discipline. The authors categorised their submissions into one or more of the following topics:

- | | |
|---|---|
| - Algorithms (16 papers) | - Formal Methods (4 papers) |
| - Artificial Intelligence (5 papers) | - Graphics (2 papers) |
| - Communications and Networks (12 papers) | - High Performance Computing (3 papers) |
| - Compilers (2 papers) | - Human-Computer Interaction (3 papers) |
| - Computer Architecture (1 paper) | - Mobile Computing (3 papers) |
| - Computer Vision (4 papers) | - Natural Language (1 paper) |
| - Concurrency (1 paper) | - Pattern Matching and Image Processing (12 papers) |
| - Databases (6 papers) | - Programming Languages (2 papers) |
| - Data structures (2 papers) | - Reliability (4 papers) |
| - Distributed Systems (11 papers) | - Security (10 papers) |
| - E-Commerce (4 papers) | - Software Engineering (15 papers) |
| - Education (2 papers) | - Theory (2 papers) |
| - Embedded Systems (1 paper) | - Visualization (4 papers) |

The programme committee consisted of 33 highly regarded academics from around the globe, including Australia, Brazil, Canada, Japan, New Zealand, Singapore and USA. All papers were sent to at least three programme committee members for review and every effort was made to obtain at least three reviews. Of the 67 papers submitted, 24 were selected for presentation at the conference.

The programme committee invited Dr Genevieve Bell, to give a keynote on *Peeping, flashing and happy slapping: the future(s) of computing*. Dr Bell is a cultural anthropologist and director of Intel's Domestic Design and Technology Research in Portland, Oregon. The committee also invited Professor Jenny Edwards, Professor Leon Sterling and Professor Justin Zobel to give invited talks. Professor Edward's talk was entitled *Whither ICT in the RQF and the NZ PBRF?* Professor Sterling's talk was entitled *The Music Industry and P2P networks: From enemies to collaborators* and Professor Zobel's talk was entitled *The Case of the Duplicate Documents: Measurement, Search, and Science*.

We thank all authors who submitted papers and all conference participants for helping to make the conference a success. We also thank the members of the programme committee and the external referees for their expertise in carefully reviewing the papers. We are grateful to Professor John Roddick for his assistance in the production of the proceedings and Sharon Liu for her work in managing the reviewing system and processes. We thank Professor Jenny Edwards for her support as the President of CORE (Computing Research and Education Association of Australasia). Last, we express our gratitude to our hosts in Ballarat.

Gillian Dobbie
University of Auckland

ACSC 2007 Programme Chair
January 2007

Programme Committee

Chairs

Gillian Dobbie, University of Auckland

Members

Hussein A. Abbass, UNSW@ADFA (Australia)
Michael H. Albert, University of Otago (New Zealand)
Stephane Bressan, National University of Singapore (Singapore)
Andrew P. Bernat, Computing Research Association (USA)
Fred Brown, The University of Adelaide (Australia)
Kris Bubendorfer, Victoria University of Wellington (New Zealand)
Neville Churcher, University of Canterbury (New Zealand)
Sally Jo Cunningham, University of Waikato (New Zealand)
Trevor Dix, Monash University (Australia)
Jin Song Dong, National University of Singapore (Singapore)
Vladimir Estivill-Castro, Griffith University (Australia)
Colin Fidge, Queensland University of Technology (Australia)
Ken Hawick, Massey University - Albany, (New Zealand)
Nigel Horspool, University of Victoria (Canada)
Michael Houle, National Institute for Informatics (Japan)
Chris Johnson, The Australian National University (Australia)
Paddy Krishnan, Bond University (Australia)
Xuemin Lin, University of New South Wales (Australia)
Bernard Mans, Macquarie University (Australia)
Chris McDonald, The University of Western Australia (Australia)
Mirka Miller, University of Ballarat (Australia)
Michael Oudshoorn, Montana State University (USA)
Leon Sterling, University of Melbourne (Australia)
Masahiro Takatsuka, The University of Sydney (Australia)
Bruce H. Thomas, University of South Australia (Australia)
Geoff West, Curtin University of Technology (Australia)
Hua Wang, University of Southern Queensland (Australia)
Lyndon While, The University of Western (Australia)
Graham Williams, University of Canberra (Australia)
Burkhard Wuensche, University of Auckland (New Zealand)
Yanchun Zhang, Victoria University (Australia)
Avelino Zorzo, Pontificia Universidade Catolica do Rio Grande do Sul (Brazil)

Additional Reviewers

Andrew R. Bernat
Anne Boettcher
Shane Bracher
Chunqing Chen
Alan Colman
Yuzhang Feng
Kendall Lister
Christof Lutteroth
Tariq Mahmood
Jing Sun
Jun Sun
Warren Toomey
Zheng da Wu

Organising Committee

Welcome

I would like to welcome you to the University of Ballarat and ACSW 2007.

Ballarat is one of Australia's largest inland cities with a population of 83,000, and is nestled peacefully in the heart of Victoria just over an hour from Melbourne. Ballarat is regarded as the birthplace of democracy in Australia and has one of the finest tourist attractions, namely Sovereign Hill.

The University of Ballarat is the third oldest tertiary institution in Australia. It is a medium-size University with about 22,000 students. The School of Information Technology and Mathematical Sciences of the University of Ballarat has 80 academic and general staff and includes the research Centre for Informatics and Applied Optimization (CIAO) and the Collaborative Centre for eHealth (CCeH).

ACSW 2007 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 Knowledge Management and Discovery (HKMD),
- Australasian Information Security Workshop: Privacy Enhancing Systems (AISW), and the
- Australasian Computing Doctoral Consortium (ACDC).

I thank all those who have worked to ensure the success of ACSW2007 including the Organizing Committee, the Conference Chairs and Programme Committees, the invited speakers and the delegates.



Professor Sid Morris

Head, School of Information Technology and Mathematical Sciences
University of Ballarat
January, 2007

General Chair

Professor Sid Morris, School of Information Technology and Mathematical Sciences, University of Ballarat

Organising Committee Members

Ms Nadine Gass
Mr Sasha Ivkovic
Ms Kathleen Keogh
Dr Liping Ma
Dr Prabhu Manyem
Mr Greg Simmons
Ms Rosemary Torney
Dr Chris Turville
Ms Belinda Wallesz
Dr David Yost

CORE - Computing Research and Education

CORE welcomes all delegates to ACSW2007 in Ballarat.

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.

2006 has been a difficult year for IT and especially CORE's members. Falling student numbers have meant cutbacks in many of our universities. However, despite offshoring, industry is now calling for more graduates. We'll continue to work with ACS and industry bodies to try to convey this message to school leavers and their parents. We also have to continue to impress on industry the need to provide entry level positions so that young people can work towards the more senior positions which are so understaffed.

RQF is still hovering over Australian universities. In preparation, CORE has been part of a major exercise this year to rank ICT conferences and we'll contribute to the work of our sister organisation, ACPHIS in a similar journal ranking exercise.

Thank you all for your contributions in 2006 and we look forward to an exciting 2007.

CO_{mputing}
R_{esearch}
& E_{ducation}

Jenny Edwards

President, Computing Research and Education

January, 2007

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.

2009 (Proposed). Communications Volume Number 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.

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 AUC.

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 Conference on Computing Education.
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.

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

ACSW and ACSC 2007 Sponsors

We wish to thank the following sponsors for their contribution towards this conference. For an up-to-date overview of sponsors of ACSW 2007 and ACSC 2007, please see <http://www.ballarat.edu.au/acsw/>.



University of Ballarat, Australia



**AUSTRALIAN
COMPUTER
SOCIETY**

Australian Computer Society



CORE - Computing Research and Education



**THE UNIVERSITY OF AUCKLAND
NEW ZEALAND**

Department of Computer Science

KEYNOTE

Just Like Magic: Anthropological Musings on the History and Culture of Wireless Technologies

Genevieve Bell

Digital Home Group, Intel
20270 NW AmberGlen Ct., AG1-112
Beaverton, OR 97006 , USA

In 1840, Maori elders from New Zealand's north island agreed to the terms and conditions of a British treaty. Amongst its many provisions, the Treaty of Waitangi as it is commonly known, retained Maoris rights in land and taonga (treasures). In 2005, Nextscribe.org, a Catholic think-tank in New Mexico declared that the “network is the church” and set out an ambitious agenda for research into the role that technology might play in the spiritual lives of America's (and the world's) Catholics. What do these events have in common, and why might they be relevant to our contemporary discussions about wireless technologies?

In this talk, I propose to re-examine the notion of 'wirelessness' from an anthropological perspective. This paper is informed by nearly a decade of ethnographic research, with a particular focus on the Asia region, and by ethnographic and feminist theory. I draw on historical and contemporary cultural practices, events and accounts to create 5 interpretative frames for wirelessness. Wireless as schematics; practice(s); politics; citizenship; and imagined. Using these frameworks I suggest a different way of thinking about one of the dominant technology infrastructures of this decade.

Copyright (c) 2007, Australian Computer Society, Inc.
This paper appeared at the Thirtieth Australasian Computer Science Conference (ACSC2007), Ballarat, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 62. Gillian Dobbie, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

CONTRIBUTED PAPERS

Compact Layout of Layered Trees

Kim Marriott

Peter Sbarski

Clayton School of IT, Monash University, Australia.
 {marriott,sbarski}@mail.csse.monash.edu.au

Abstract

The standard layered drawing convention for trees in which the vertical placement of a node is given by its level in the tree and each node is centered between its children can lead to drawings which are quite wide. We present two new drawing conventions which reduce the layout width to be less than some maximum width while still maintaining the essential layered drawing convention. These conventions relax the requirement that a parent must be exactly placed midway between its children, and instead make this a preference which can be violated if this is required for the layout to fit into the required width. Both drawing conventions give rise to a simple kind of quadratic programming problem. We give an iterative gradient projection algorithm for solving this kind of problem, and also a linear time heuristic algorithm. Our algorithms are practical: a tree with three thousand nodes can be laid out in less than a hundred milliseconds with either algorithm.

1 Introduction

Trees, i.e. connected acyclic graphs, occur in a wide variety of applications, including computer data structures, parse trees, hierarchical database models, phylogenetic trees, hierarchically organised file systems (for example, directories and files), decision trees, organization charts, family trees and biological classifications. Given the practical importance of trees it is not surprising that there has been considerable research into tree layout. The standard layered drawing convention of trees stipulates (Brüggermann-Klein & Wood 1989, Kennedy 1996) that:

1. the y -coordinate of each node corresponds to its level,
2. nodes on the same level are separated by a minimum gap,
3. edges do not cross each other,
4. each node is centered directly over its children,
5. the drawing is symmetrical with respect to reflection,
6. the trees are drawn compactly.

Wetherell and Shannon (Wetherell & Shannon 1979) proposed a linear time algorithm for layered drawing of binary trees, and this was improved by

Reingold and Tilford (Reingold & Tilford 1981). Walker (Walker 1990) extended the Reingold-Tilford algorithm to handle n -ary trees, while Buchheim, Jünger and Leipert (Buchheim, Jünger & Leipert 2002) gave an improved version of Walker's algorithm with linear time. The Reingold-Tilford algorithm and the subsequent extensions are the standard methods for drawing rooted ordered trees.

However, a disadvantage of these algorithms is that the resulting drawings can be quite uncompact as the real-world example in Figure 1 shows. In part for this reason, a number of other drawing conventions for trees have been suggested (di Battista, Eades, Tamassia & Tollis 1999). These include radial drawings in which the layers are mapped to concentric circles and hv-drawings in which the edges are either drawn horizontally to the right, or vertically to the bottom. The disadvantage of these other drawing conventions is that the structure and hierarchical nature of the tree is less clear, and the layout looks "unnatural".

We present two new drawing conventions which reduce the layout width to fit in some maximum drawing width while still maintaining the essential layered drawing convention. Both conventions relax the requirement that a parent must be exactly placed at the midway between its children, and instead make this a preference which can be relaxed if this is required for the layout to fit into some maximum width. The drawing conventions give rise to a simple kind of quadratic programming problem in which a quadratic objective must be minimized subject to minimum separation between nodes on each level, and the layout fitting in the maximum width. We give a gradient projection algorithm for solving this kind of problem, and also a linear time heuristic algorithm.

Our techniques have application to most real-world visualisation of trees since a significant disadvantage of the standard layered drawing convention is that even for moderate sized trees the drawing becomes quite large and cannot easily be viewed on a computer monitor or printed on a reasonable number of pages. Thus there is a need to find layouts which are more compact and which better utilise the space in a page or on a screen. Figures 2, 3 and 4 show examples of the layout produced by our algorithms for the example from Figure 1.

In the next section we review the standard algorithms for layout of layered trees. In Section 3 we present the new drawing conventions, Section 4 contains the algorithms for finding a layout satisfying the conventions, and Section 5 gives an empirical evaluation of their performance, while Section 6 concludes.

2 Background

The Reingold-Tilford algorithm (Reingold & Tilford 1981) uses a divide-and-conquer strategy to find a lay-

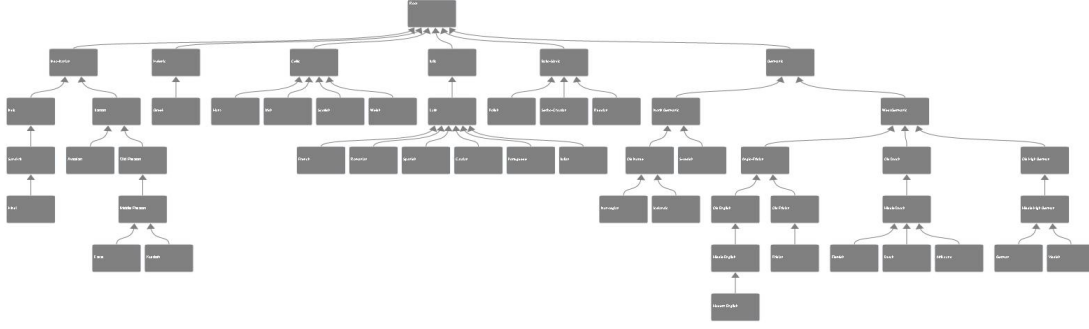


Figure 1: Example tree detailing the relationship between Indo-European languages drawn with the Walker Algorithm.

out. A single node is placed at 0, and a non-leaf node i is placed by a recursively laying out its sub-trees and then squashing the sub-trees as close as possible together and then placing i midway between the leftmost and rightmost child. Clever use of data structures allows this to be done in linear time.

Reingold-Tilford's algorithm can be modified to handle n -ary trees by traversing the children of a node left to right, placing and shifting the corresponding subtrees one after another. However, this approach may violate the aesthetic goal that the tree must be symmetric with respect to reflection since some branches can end up being clustered closely together as they are placed as close as possible to the left rather than being evenly spaced apart. Walker (Walker 1990) gave a modification of Reingold-Tilford's algorithm for n -ary tree layout which overcomes this problem. The contours of a node's sub-trees are traversed, and if there is an overlap, the right subtree is shifted by the minimum amount to the right. Any smaller subtrees situated between the overlapping left and right subtree are uniformly spaced between the two outer sub-trees. This guarantees that the symmetry is preserved. Unfortunately, the algorithm given by Walker has quadratic worst case complexity. Buchheim, Jünger and Leipert (Buchheim et al. 2002) improved Walker's algorithm to have linear time.

One nice property of Reingold-Tilford's and Walker's approach is that the drawing of a sub-tree does not depend on its context, and so isomorphic subtrees will have the same layout, and that drawings are symmetric with respect to reflection. However, because of this it will not always produce a layout of minimum width, as minimising the over all width may require that some sub-trees are drawn non-compactly. The problem is that minimizing the total width may mean that isomorphic subtrees need to be laid out differently in different parts of the tree as their global context is important.

If we do wish to minimize the total width while still placing a parent midway between its children then this can be modelled as a linear programming problem and solved in polynomial time (Supowit & Reingold 1983). However, in practice, the requirement that the parent is exactly midway between its children means that the minimal width layout is only very rarely narrower than that found using Reingold-Tilford's and Walker's approach. Therefore, in the next section we investigate drawing conventions in which the requirement to place each parent exactly midway between its children is relaxed. As we shall see this allows narrower layout.

3 Drawing Conventions that Relax Centering of Parents

Requiring that parents are midway between their children means that the drawing cannot be as narrow as possible. In this section we give two new drawing conventions for layered trees in which we must find the best layout for the tree that is no wider than some fixed width W . W might be set by the user interactively to try and better fit the layout within a display or print region. Of course, we require that W is greater than the cumulative width of the nodes on each level, since otherwise there is no drawing in which the nodes do not overlap.

We assume there are n nodes, $1, \dots, n$ where node 1 is the root of the tree, and $1, \dots, m$ levels with the root on level 1. We let w_i give the width of node i , and $rt(i)$ be the index of the node to the immediate right of i on the same level and let gap_i give the minimum gap between i and the node to its immediate right. For each level j , $lm(j)$ and $rm(j)$ respectively give the index of the leftmost and rightmost node on that level. We assume that nodes are numbered consecutively on each level.

We let x_i be the horizontal position of node i . We let $par(i)$ be the index of the parent of i , and $lc(i)$ and $rc(i)$ the index of the leftmost and rightmost child of i (they are the same if i has one child). We use the convention that if the node referred to by indexing function does not exist then the indexing function returns 0: For instance, $par(1)$ returns 0 as the root of the tree has no parent.

The first drawing convention is based on making trees compact by minimizing edge lengths. This has the affect of placing nodes at the average of their parent and children's positions subject to the non-overlap and minimum width constraints.

Drawing Convention: Minimizing distance between parents and their children (Min-Dist)

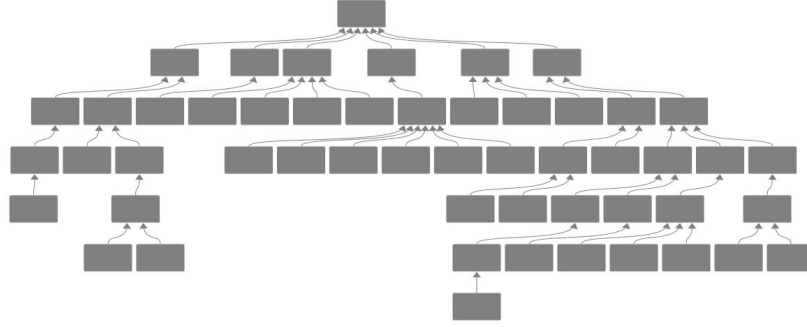
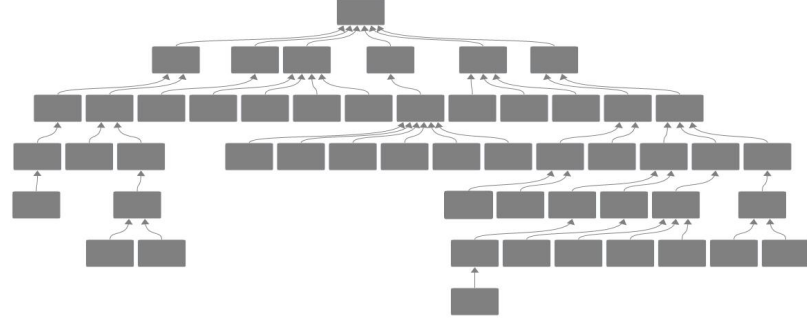
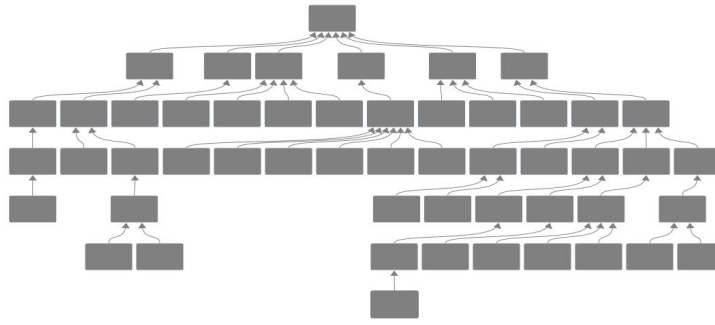
Minimize

$$\sum_{i=2}^n (x_{par(i)} - x_i)^2$$

subject to: for all $i \in 2, \dots, n$

- $x_i + \frac{w_i}{2} + gap_i \leq x_{rt(i)} - \frac{w_{rt(i)}}{2}$, if $rt(i) \neq 0$
- and for all $j \in 1, \dots, m$,
- $0 \leq x_{lm(j)} - \frac{w_{lm(j)}}{2}$
- $x_{rm(j)} + \frac{w_{rm(j)}}{2} \leq W$.

This is similar to the objective used in Gansner et al. (Gansner, Koutsofios, North & Vo 1993) to determine the placement of nodes on each layer in a Directed Acyclic Graphs (DAGs) drawn using the Layered drawing convention. Given a horizontal ordering

(a) Min-Dist: $W = W_{max}$ (b) Min-Dist: $W = W_{mid}$ (c) Min-Dist: $W = W_{min}$ Figure 2: Example tree from Figure 1 drawn with the Min-Dist drawing convention for different maximum widths W using the gradient projection algorithm.

of the nodes on each layer, their algorithm determines the x -coordinates by minimizing edge lengths between nodes. The main difference is that they model this as a linear problem and so minimize

$$\sum_{i=2}^n |x_{par(i)} - x_i|$$

rather than

$$\sum_{i=2}^n (x_{par(i)} - x_i)^2.$$

As pointed out in (Marriott, Moulder, Hope & Twardy 2005), using a linear objective may lead to unnecessary asymmetry as parents are not necessarily centred with respect to their children since the objective function uses absolute value. Thus, if the root of the tree has two children, it can be placed at any point between these without changing the penalty. Therefore, we prefer to use a quadratic objective function since in the above example this will prefer to place the root midway between its children.

However, in general the Min-Dist drawing convention will not place nodes midway between their children but rather at the average of their children

and parent's horizontal position, since this minimizes the total horizontal extent of the edges. If placing a parent exactly between its leftmost and rightmost child is regarded as important we can extend the Min-Dist drawing convention by adding another term to the objective function to achieve this:

Drawing Convention: Placing parents midway between their children (Par-Midway)
Minimize

$$\sum_{i=2}^n (x_{par(i)} - x_i)^2 + \alpha \times \sum_{\substack{1 \leq i \leq n \text{ s.t.} \\ rc(i) \neq 0}} \left(x_i - \frac{x_{lc(i)} + x_{rc(i)}}{2} \right)^2$$

subject to: for all $i \in 2, \dots, n$

- $x_i + \frac{w_i}{2} + gap_i \leq x_{rt(i)} - \frac{w_{rt(i)}}{2}$, if $rt(i) \neq 0$
- and for all $j \in 1, \dots, m$,
- $0 \leq x_{lm(j)} - \frac{w_{lm(j)}}{2}$
- $x_{rm(j)} + \frac{w_{rm(j)}}{2} \leq W$.

The scaling coefficient α specifies the relative importance of the two components of the objective. If α is equal to 0, then Par-Midway is identical to the Min-Dist drawing convention, but as α increases in

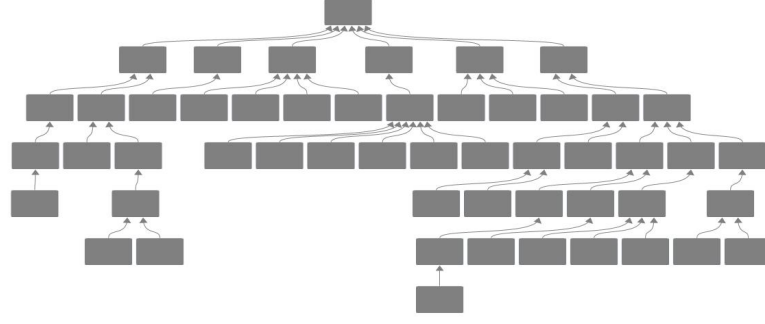
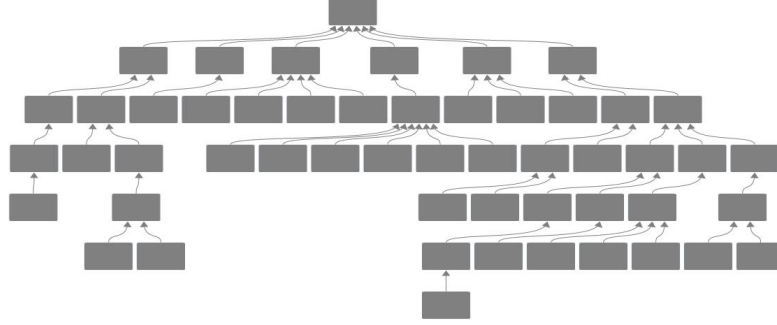
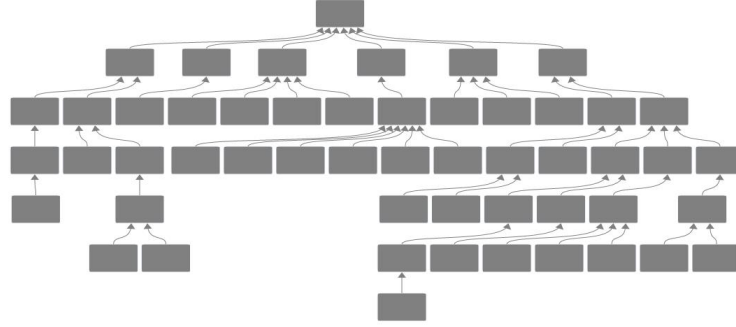
(a) Par-Midway: $W = W_{max}, \alpha = 1.0$ (b) Par-Midway: $W = W_{mid}, \alpha = 1.0$ (c) Par-Midway: $W = W_{min}, \alpha = 1.0$

Figure 3: Example tree from Figure 1 drawn with the Par-Midway drawing convention for different maximum widths W and a small value of α using the gradient projection algorithm.

value the resulting layout will tend to place parents midway between their children. It is also worth pointing out that if W is wider than the width of the layout obtained with Walker, then the layout obtained with Par-Midway for large α will be very similar to that obtained by the Walker algorithm. Of course, the main feature of interest is that W can be decreased down to the minimum width for the tree, and the two drawing conventions will “squish” the layout into that width.

Figures 2, 3 and 4 show examples of the layout produced by our algorithms for the example from Figure 1 and illustrate the effect of α and W on the layout. We give the layout for three widths: W_{max} , the width of the layout using the Walker algorithm; W_{mid} , the width of the layout of the narrowest possible layout; and W_{min} , the average of W_{max} and W_{min} . We can see that the Min-Dist drawing convention always gives quite compact, narrow layout.

4 Layout Algorithms

In this section we investigate how to find layouts which satisfy the Par-Midway and Min-Dist drawing conventions. Both require minimising a quadratic ob-

jective function of form

$$\min_x x^T A x$$

subject to some separation constraints between nodes on the same level where a *separation constraint* c is of form $u + a \leq v$ where u, v are variables and a is the minimum gap between them or an upper or lower bound on a variable v of form $c \leq v$ or $v \leq c$.

For the Min-Dist drawing convention we have that A is A^{MD} where for each node i ,

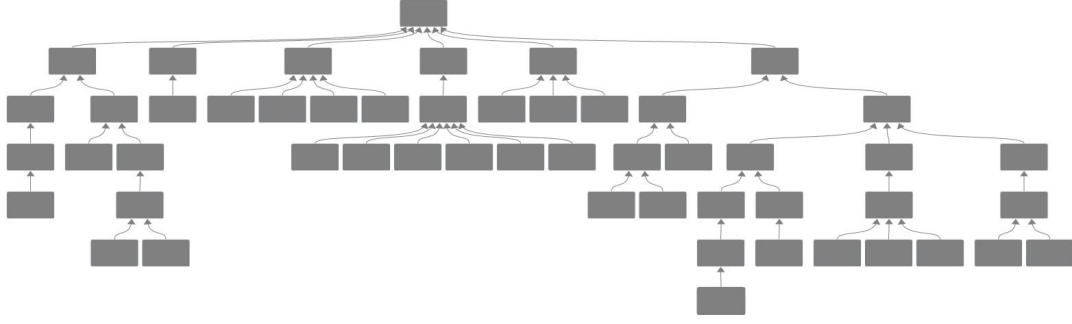
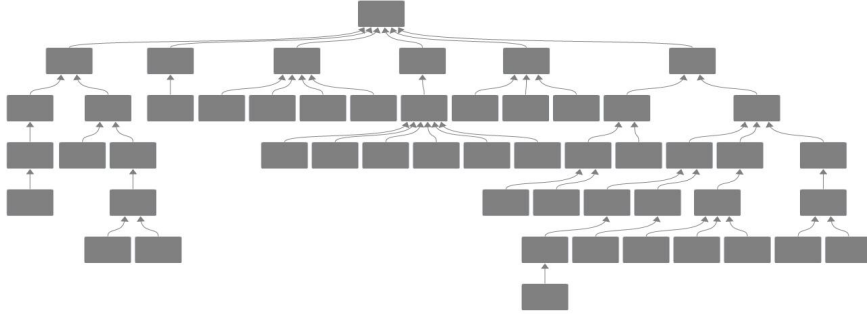
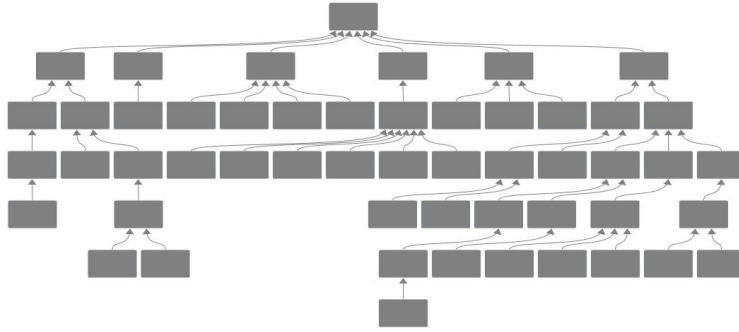
- $A_{i,i}^{MD} = \deg(i)$ (where the degree, $\deg(i)$, of i is the number of children of i plus the number of parents) and
- $A_{i,par(i)}^{MD} = A_{par(i),i}^{MD} = -1$

and all other entries in A^{MD} are zero.

For the Par-Midway drawing convention we have that A is

$$A^{PM} = A^{MD} + \alpha \times \sum_{i=1}^n C^i$$

where for each node i ,

(d) Par-Midway: $W = W_{max}, \alpha = 1.0E7$ (e) Par-Midway: $W = W_{mid}, \alpha = 1.0E7$ (f) Par-Midway: $W = W_{min}, \alpha = 1.0E7$ Figure 4: Example tree from Figure 1 drawn with the Par-Midway drawing convention for different maximum widths W for a very large α using the gradient projection algorithm.

- if i has no children, $C^i = 0$,
- if i has one child c , non-zero entries are

$$C_{i,i}^i = C_{c,c}^i = 1,$$

$$C_{i,c}^i = C_{c,i}^i = -1$$
- if i has leftmost and rightmost children l and r , the non-zero entries are

$$C_{i,i}^i = 1,$$

$$C_{l,l}^i = C_{r,r}^i = 1/4,$$

$$C_{i,l}^i = C_{l,i}^i = C_{i,r}^i = C_{r,i}^i = -1/2,$$

$$C_{l,r}^i = C_{r,l}^i = 1/4.$$

For both drawing conventions the matrix A is symmetric and positive semi-definite. Since separation constraints are linear this means that both conventions require solving a convex quadratic program.

4.1 Iterative Gradient Projection Methods

We now give an iterative *gradient-projection* algorithm (see Bertsekas (Bertsekas 1999)) for finding a layout for the above kind of problem. It is based on

the gradient projection algorithm given in (Dwyer, Koren & Marriott 2006) but specialized to the particular case of trees. The algorithm, *gp_layout*, is shown in Fig. 5. This works by taking an initial layout, such as that obtained with the Walker algorithm, and then iteratively improving the placement of nodes. At each iteration, the direction of steepest descent, $-g$, is computed where g is the gradient $\nabla x^T A x = 2Ax$ (actually $\frac{1}{2}g$ is computed, but multiplying the descent direction by a constant has no affect). Then the step size s along $-g$ from x that minimizes the objective function is determined and x is set to this new position. Since the new value of x may violate the separation constraints this is corrected by calling *project*, which returns the closest point \bar{x} to x which satisfies the separation constraints, i.e. it projects x on to the feasible region. Finally, a vector d from the initial position \hat{x} to \bar{x} is computed and a decrease in the objective when moving in this direction is ensured by computing a second stepsize α which minimizes the objective in this interval.

While the algorithm given in Figure 5 describes a fairly standard gradient-projection approach, the procedure *project* is specific to our particular quadratic program. The main difficulty in implementing gradient-projection methods is the need to efficiently

```

procedure gp_layout( $A, W$ )
   $x \leftarrow \text{initial\_soln}()$ 
  repeat
     $g \leftarrow Ax$ 
     $s \leftarrow \frac{g^T g}{g^T A g}$ 
     $\hat{x} \leftarrow x$ 
     $x \leftarrow \hat{x} - sg$ 
     $\bar{x} \leftarrow \text{project}(x, W)$ 
     $d \leftarrow \bar{x} - \hat{x}$ 
     $\alpha \leftarrow \min(-\frac{g^T d}{d^T A d}, 1)$ 
     $x \leftarrow \hat{x} + \alpha d$ 
  until  $\|\hat{x} - x\|$  sufficiently small
  return  $x$ 

procedure project( $x, W$ )
  for  $j \leftarrow 1$  to  $m$  do
     $g \leftarrow [gap_i \mid i \in lm(j), \dots, rm(j) - 1]$ 
     $d \leftarrow [x_i \mid i \in lm(j), \dots, rm(j)]$ 
     $x_{lm(j)}, \dots, x_{rm(j)} \leftarrow \text{level\_project}(d, g, W)$ 
  end for
  return  $x$ 

```

Figure 5: Gradient projection algorithm to layout a tree with n nodes on m levels s.t. the layout is no wider than W , nodes have a minimum separation and the position x for the nodes minimizes $x^T A x$ where A is a symmetric positive-semidefinite matrix.

project on to the feasible region. That is, we must solve the quadratic problem

$$\min_y \sum_{i=1}^n (y_i - d_i)^2$$

subject to the same constraints on the variables y as the original problem.

Fortunately, the procedure *optimal_layout* given in (Marriott et al. 2005) can be used to project on to the feasible region. It solves the quadratic problem

$$\min_y \sum_{i=1}^m w_i \times (y_i - d_i)^2$$

subject to a separation constraint of form $y_i + g_i \leq y_{i+1}$ for each $i = 1, \dots, m-1$ where g_i is the minimum separation between y_i and y_{i+1} , d_i is the desired position for y_i and w_i the importance of placing y_i at d_i .

The algorithm works by merging variables into larger and larger blocks of variables where a block is a sequence of variables with the minimum gap between each pair. The variables are processed left to right. When variable y_i is processed it is placed in a new block b at its desired location d_i . If the preceding block overlaps b it is merged with b and the desired position of the block is set to the weighted mean of the desired locations of variables in the block. This is repeated until b does not overlap the preceding block. The algorithm has linear complexity.

We can use *optimal_layout* to compute the projection for the nodes in a single layer: the only trick is that we add a dummy node at the beginning of the layer and one at the end whose desired values are 0 and W respectively, and whose weight is much higher than that of the other nodes. The two dummy nodes force the other nodes to fit in the desired width for the layout. The procedure *level_project* does this. By combining the projection for each layer, we have the overall projection. Since *optimal_layout* has linear time complexity, the overall projection has linear time complexity.

```

procedure bottom_up_narrow( $W$ )
   $x \leftarrow \text{walker}()$ 
  for  $j \leftarrow m$  to 1 do
     $g \leftarrow [gap_i \mid i \in lm(j), \dots, rm(j) - 1]$ 
     $d \leftarrow [des(i) \mid i \in lm(j), \dots, rm(j)]$ 
     $x_{lm(j)}, \dots, x_{rm(j)} \leftarrow \text{level\_project}(d, g, W)$ 
  end for
  return  $x$ 

```

Figure 7: Linear time bottom-up algorithm to layout a tree with n nodes on m levels s.t. the layout is no wider than W and nodes have a minimum separation.

It follows from standard results on gradient projection and the fact that this is a convex optimisation problem that

Theorem 1 *gp_layout converges to a solution that minimizes $x^T A x$ subject to the node separation and maximum width constraints.*

As we shall investigate in the next section, the algorithm is reasonably fast. The matrix A has only $O(n)$ non-zero entries. This means that as long as a sparse representation is used for A , each iteration takes only linear time since the projection step takes only linear time.

4.2 Bottom-up Algorithm

We have also investigated a one pass bottom-up approach to find a layout satisfying the width restriction. The algorithm is given in Fig. 7. The idea is quite simple. We first determine a layout using Walker's algorithm. Now we process the layers from the bottom up. For each layer j , we compute the desired position d_i of each node i in the layer using the function *des*(i) where if i has children, *des*(i) is their midpoint $(x_{lc(i)} + x_{rc(i)})/2$, otherwise *des*(i) is the node's current position x_i . The new position, x_i , for each node i is computed by using *level_project* to project the desired values on to the closest solution satisfying the minimum width and node separation constraints for that level, effectively squishing them into the width W .

Figure 6 shows the layout for various values of W for the examples from Figure 1. It is instructive to compare them with the layouts shown in Figures 2, 3 and 4 obtained using the gradient projection algorithm to find a layout satisfying the Par-Midway drawing convention for $\alpha = \infty$, since the procedure *bottom_up_narrow* can be viewed as a heuristic for finding a layout satisfying this convention. As these examples show, it is not guaranteed to find the optimal solution, but, it will always find a solution satisfying the width and separation constraints, and it does this in linear time assuming the linear time version of Walker's algorithm is used.

5 Evaluation

To evaluate the efficiency of our algorithms we laid out 10 random trees, with 100 to 3000 nodes. Times can be seen in Table 5.

All experiments were run on a 1.83 GHz Intel Centrino with 1GB of RAM. The algorithms were implemented using Microsoft Visual C# Compiler version 8.00.50727.42 for Microsoft .NET Framework version 2.0.50727 under the Windows XP Service Pack 2 operating system. We used the original Walker algorithm

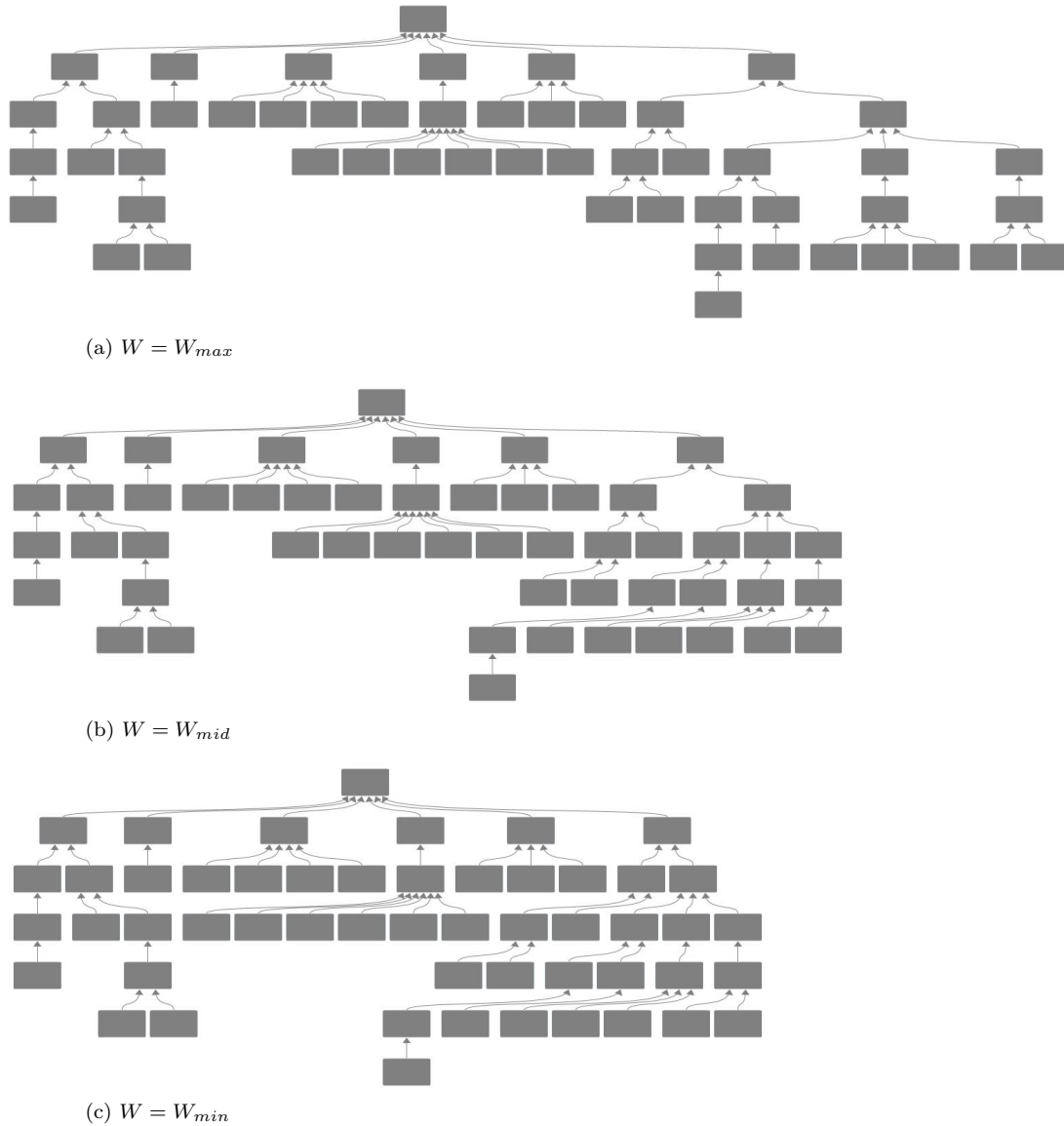


Figure 6: Example tree from Figure 1 drawn for different maximum widths W using the Bottom-up Layout Algorithm.

rather than the linear version: as the results show performance of the original algorithm is effectively linear and extremely fast.

We used our implementation of the Walker algorithm to compute the initial solution for both the bottom-up and the gradient projection algorithms. We did not include the time taken to compute this initial solution: this is simply the time for running Walker's algorithm. The tolerance for terminating the gradient projection algorithm was set to 0.002 of the width.

The speed of both the gradient projection algorithm and the bottom-up algorithm is very satisfying: even very large trees can be laid out in less than 200 milliseconds and the speed is comparable with that of Walker's algorithm, the standard layered tree layout algorithm. The main reason for the fast performance of the gradient projection algorithm is that the number of iterations is quite small since the algorithm quickly converges to the optimal layout.

6 Conclusion

The standard layered drawing convention for trees can lead to wide layouts because each parent must be placed exactly midway between its children. We

have introduced two new drawing conventions for layered trees: *Min-Dist* in which the horizontal distance between parent's and children is minimised, and *Par-Midway* in which we minimize the horizontal distance between parent's and their children but also try and minimise the distance between a parent and the midpoint of its children. Thus both drawing conventions relax the requirement that a parent must be exactly placed at the center of its children, and instead effectively make this a preference which can be relaxed if this allows the tree to fit in a drawing of the desired width.

We have given an iterative gradient projection based algorithm for computing tree layouts using these two new drawing conventions. Experimental evaluation shows that the algorithm is fast enough for practical applications: a tree with three thousand nodes can be laid out in a few hundred milliseconds, which is only a few times slower than Walker's algorithm. We have also given a linear-time bottom-up algorithm for narrowing a layout produced by the Walker algorithm. This is somewhat faster, with similar speed to Walker's algorithm, and takes less than 100 milliseconds to layout a tree with over three thousand nodes. The layout quality is also quite good and similar to that of the gradient projection algorithm.

Nodes	Levels	Walker	Min-Dist		Par-Midway ($\alpha = 1.0$)		Bottom-up	
			W_{max}	W_{min}	W_{max}	W_{min}	W_{max}	W_{min}
32	9	18	16 (4)	17 (5)	15 (3)	15 (3)	4	4
49	9	18	18 (4)	20 (3)	14 (3)	16 (3)	4	4
52	7	20	18 (2)	25 (19)	16 (3)	17 (3)	4	4
84	16	21	19 (2)	21 (22)	17 (3)	21 (24)	5	4
220	16	21	43 (19)	69 (28)	39 (16)	70 (29)	6	5
365	28	22	17 (2)	25 (5)	17 (2)	29 (6)	7	7
673	22	24	29 (3)	35 (4)	20 (3)	27 (4)	11	11
1102	201	30	145 (38)	160 (40)	147 (38)	157 (39)	14	14
2101	31	33	66 (3)	79 (3)	63 (3)	65 (4)	26	21
3278	101	36	80 (3)	92 (3)	83 (3)	86 (3)	34	31

Table 1: Performance figures for 10 random trees. All timings are given in milliseconds. Figures in brackets specify the number of iterations. The gradient projection algorithm was used to compute the layout for the Min-Dist and Par-Midway drawing conventions.

References

- Bertsekas, D. P. (1999), *Nonlinear Programming*, 2nd edn, Athena Scientific.
- Brüggermann-Klein, A. & Wood, D. (1989), ‘Drawing trees nicely with TeX’, *Electronic Publishing* **2**(2), 101–115.
- Buchheim, C., Jünger, M. & Leipert, S. (2002), Improving Walker’s algorithm to run in linear time., in ‘Graph Drawing’, pp. 344–353.
- di Battista, G., Eades, P., Tamassia, R. & Tollis, I. (1999), *Graph drawing: algorithms for the visualisation of graphs*, Prentice Hall.
- Dwyer, T., Koren, Y. & Marriott, K. (2006), IPSEP-COLA: An incremental procedure for separation constraint layout of graphs, in ‘Proc. IEEE Symp. on Information Visualisation (Infovis’06)’.
- Gansner, E. R., Koutsofios, E., North, S. C. & Vo, K.-P. (1993), ‘A technique for drawing directed graphs’, *IEEE Transactions on Software Engineering* **19**(3), 214–230.
- Kennedy, A. J. (1996), ‘Drawing trees’, *Functional Programming* **6**(3), 527–534.
- Marriott, K., Moulder, P., Hope, L. & Twardy, C. (2005), Layout of Bayesian networks, in ‘Australian Computer Science Conference’, Vol. 38.
- Reingold, E. M. & Tilford, J. S. (1981), ‘Tidier drawings of trees’, *IEEE Transactions on Software Engineering* **7**(2), 223–228.
- Supowit, K. & Reingold, E. (1983), ‘The complexity of drawing trees nicely’, *Acta Informatica* **18**.
- Walker, J. Q. (1990), ‘A node-positioning algorithm for general trees’, *Software Practice and Experience* **20**(7), 685–705.
- Wetherell, C. & Shannon, A. (1979), ‘Tidy drawings of trees’, *IEEE Transactions on Software Engineering* **5**(5), 514–520.

Improved Shortest Path Algorithms for Nearly Acyclic Directed Graphs

Lin Tian

Tadao Takaoka

Department of Computer Science and Software Engineering
University of Canterbury, Christchurch, New Zealand,
Email: lti15@student.canterbury.ac.nz (Lin Tian),
Email: tad@cosc.canterbury.ac.nz (Tadao Takaoka)

Abstract

This paper presents new algorithms for computing single source shortest paths (SSSPs) in a nearly acyclic directed graph G . The first part introduces *higher-order decomposition*. This decomposition is an extension of the technique of strongly connected component (sc-component) decomposition. The second part presents a new method for measuring acyclicity based on modifications to two existing methods. In the new method, we decompose the graph into a 1-dominator set, which is a set of acyclic subgraphs where each subgraph is dominated by one trigger vertex. Meanwhile we compute sc-components of a de-generated graph derived from triggers. Using this preprocessing, a new SSSP algorithm has $O(m + r \log l)$ time complexity, where r is the size of the 1-dominator set, and l is the size of the largest sc-component. In the third part, we modify the concept of a 1-dominator set to that of a 1-2-dominator set. Each of acyclic subgraphs obtained by the 1-2-dominator decomposition are dominated by one or two trigger vertices cooperatively. Such subgraphs are potentially larger than those decomposed by the 1-dominator set. Thus fewer trigger vertices are needed to cover the graph.

Keywords: Algorithm; shortest paths; nearly acyclic graph; strongly-connected component; multidominator set; higher order decomposition

1 Introduction

Let $G = (V, E)$ be a directed graph, where V is the set of vertices, and E is the set of edges. Let each edge have a non-negative cost. The cost of a path is the sum of the costs of edges on the path. The single source shortest path (SSSP) problem is to find a path with minimum cost from a source vertex s to every vertex $v \in V$. Throughout this paper, any unspecified graph indicates a directed graph with positively real-valued edge costs, and all vertices are reachable from the source. Time complexities of algorithms use the worst case time complexity analysis.

For unrestricted graphs, Dijkstra's algorithm (Dijkstra 1959) is still the most efficient algorithm for the SSSP problem. When Dijkstra's algorithm uses an efficient data structure, such as Fibonacci heaps (Fredman & Tarjan 1987) or 2-3 heaps (Takaoka 2003) in its priority queue manipulations, it can achieve $O(m + n \log n)$ time where n is the number of vertices, and m is the number of edges of a given

graph. We assume that the SSSP algorithms that appear in this paper use Fibonacci heaps or 2-3 heaps for their priority queue manipulations. However, for restricted digraphs, we have efficient alternatives, like acyclic graphs with $O(m+n)$ time (Tarjan 1983) and planar graphs with $O(n\sqrt{\log n})$ time (Frederickson 1987).

If a graph is *nearly acyclic*, obviously we should not use the conventional algorithms for general graphs. If we use Dijkstra's algorithm, it does not count the underlying graph structure, and always involves n delete-min operations. In order to efficiently compute the shortest paths for nearly acyclic graphs, several specialized algorithms have been published (Abuaiadh & Kingston 1993, Abuaiadh & Kingston 1994, Takaoka 1998, Saunders & Takaoka 2003, Saunders & Takaoka 2005). These work have shown that we can reduce the number of delete-min operations performed in priority queue manipulations.

However, those specialized algorithms are based on two different measures of what a nearly acyclic graph is. (1) Takaoka gives a definition of acyclicity — the degree of cyclicity of a graph G , $cyc(G)$, is defined by the maximum cardinality of the strongly connected components (sc-components) of G . When the $cyc(G)$ is small, he categorizes the given graph as a nearly acyclic graph (Takaoka 1998). (2) Saunders states that a nearly acyclic graph is a graph that contains relatively few acyclic subgraphs, each subgraph of which is dominated by a vertex, called a trigger (Saunders 2004). Obviously, removal of triggers cuts all cycles in the graph. Saunders' idea is similar to the measure used by Abuaiadh and Kingston (1994), who say a graph is nearly acyclic if there are very few simple cycles in the graph. Note that we need preprocessing to use the above properties of near acyclicity. Here, we measure the near acyclicity of the graph by those parameters such as $k = cyc(G)$ and $r = \text{number of triggers}$. The smaller the values of the parameters are, the more acyclicity the graph has. These two measures (1) and (2) are independent and can not explain one another. We will have a more detailed review of these work in the next section.

The first part of this paper describes an extended technique of sc-components decomposition. We call it the *higher-order decomposition*. It is developed upon Takaoka's technique of strongly connected component decomposition for nearly acyclic graphs (Takaoka 1998), which we have mentioned earlier in this section. In sc-component decomposition, a graph is decomposed into sc-components. Thus, for computing the shortest paths, we run Dijkstra's algorithm only for each sc-component but not the whole graph. When all the sc-components are small, then we can efficiently solve the SSSP problem. Based on the *higher-order decomposition*, we give another definition of acyclicity. That is, the degree of cyclicity

$cyc^h(G)$ is the maximum cardinality of the strongly connected components of the decomposed graph G after the h^{th} order decomposition is made. The original definition introduced by Takaoka (1998) can be represented as: the degree of cyclicity $cyc(G)$ is the maximum cardinality of the strongly connected components of the decomposed graph G after the 1^{th} order decomposition is made.

In the second part of the paper we combine the two measures of near acyclicity into one, and show how to efficiently solve the SSSP problem in nearly acyclic graphs. For preprocessing we use a *hierarchical depth first search* (HDFS) algorithm to decompose a graph. This algorithm does 1-dominator decomposition and decomposition on triggers into sc-components at the same time. The computing time for preprocessing is $O(m)$. We degenerate the graph in such a way that each new vertex is a trigger in the original graph and a new edge exists from a vertex u to a vertex v if there is an edge from the corresponding acyclic subgraph dominated by u to v . Let r be the number of triggers and l be the maximum size of sc-components in the degenerated graph. Using this preprocessing, we show that we can efficiently solve the SSSP problem for nearly acyclic graphs with these parameters in $O(m+r\log l)$ time.

In the third part of this paper, we modify the concept of 1-dominator sets to define 1-2-dominator sets. In a 1-2-dominator set, generally speaking, one or two trigger vertices cooperatively dominate an acyclic structure in a graph. This offers potentially larger acyclic structures than the 1-dominator set does. Thereby, fewer trigger vertices are needed to cover the whole graph, that is, $r' \leq r$, where r' is the number of triggers in the 1-2-dominator decomposition, and r is the number of trigger vertices in the 1-dominator set. Considering efficient shortest path algorithms only do delete-min operations on trigger vertices, fewer trigger vertices can reduce the time for computing shortest paths. When r' is much smaller than r , we can gain efficiency in computing SSSPs for a nearly acyclic graph in $O(m + r'\log r')$ time. We present algorithms to achieve $O(m + r^2)$ time to compute the 1-2-dominator set in a graph.

In the following section, we review previous related work on nearly acyclic graphs. Then in Section 3, we describe the *higher-order decomposition* approach. In Section 4, we present a combined measure of acyclicity, and give the HDFS algorithm to implement this approach. In Section 5, we introduce the 2-dominator set, and present improved algorithms for computing 2-dominator sets. In Section 6, we give evaluations of algorithms presented in this paper. Section 7 gives some concluding remarks of this paper.

2 Review Of Related Work

Abuaiadh and Kingston (1993) suggested that the inherent complexity of the shortest path problem depends on the cycle structures of a graph as well as on its size. They gave an algorithm with $O(m+n\log t)$ time complexity, where t was the number of delete-min operations needed in the priority queue manipulations. For nearly acyclic graphs, t was expected to be small, so their algorithm could efficiently solve the SSSP problem. However, the value of t is defined by an algorithm, and had no direct relation with the static structure of the graph. Later in 1994, they introduced another algorithm with time complexity $O(m+k\log k)$, where k was the number of cycles in a graph. This was improved by Saunders and Takaoka (2005), who defined the concept of 1-dominator set.

Takaoka (1998) gave a definition of acyclicity. The degree of cyclicity of a graph G , $cyc(G)$, was de-

defined as the maximum cardinality of sc-components of G . When $cyc(G)$ was small, he defined the given digraph G to be nearly acyclic. When $cyc(G) = k$, he gave an algorithm with $O(m+n\log k)$ time complexity (Takaoka 1998). Take Figure 1 for example, the degree of cyclicity of the graph is 3, so that $k = 3$.

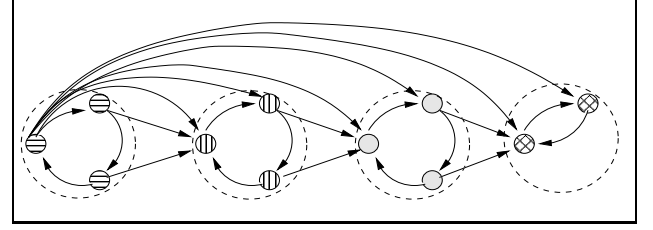


Figure 1: Vertex groups with different patterns form four sc-components in a graph. The largest sc-component has three vertices.

Obviously, Takaoka's approach needs to first compute sc-components. If $F = (V_F, E_F)$ is a depth-first spanning forest of the digraph G and $G_i = (V_i, E_i)$ is a strongly connected component, then $T_i = (V_i, E_i \cap E_F)$ is a tree (Gibbons 1985). The root vertex of the tree T_i is called *the root of the strongly connected component* G_i . If the root of an sc-component is known, then the sc-component is determined when the depth-first search from its root is finished (Gibbons 1985).

For the purpose of encoding the above statement, Algorithm 1 is an algorithm of depth-first search for sc-components by Tarjan (1972). We associate a parameter $lowlink(v)$ with each vertex v . If the vertices are labeled by variable $visitNum(v)$ according to the order in which they are visited in a depth-first search, then $lowlink(v)$ is to be the smallest of $visitNum(v)$ and those of the vertices which are connected by an edge to a descendant of v including v in the tree T_i . That is, if the depth-first search from v can reach a vertex with a lower visit number, v can not be the root of an sc-component, and $lowlink(v)$ keeps a trace of some smaller number of vertices reachable from v (Tarjan 1972).

Algorithm 1. The Depth-First Search for Strongly Connected components algorithm.

```

1. procedure CONNECT( $v$ )
2. {
3.    $visited[v] \leftarrow True$ ;  $visitNum[v] \leftarrow cnt$ ;
4.    $cnt \leftarrow cnt + 1$ ;  $lowlink[v] \leftarrow visitNum[v]$ ;
5.    $T \leftarrow T + v$ ;
6.   for all  $w \in OUT(v)$  do {
7.     if  $visited[w] = False$  then do {
8.       CONNECT( $w$ );
9.        $lowlink[v] \leftarrow \min\{lowlink[v], lowlink[w]\}$ ;
10.    } ;
11.   else if  $visitNum[w] < visitNum[v]$ 
12.     and  $w \in T$  then
13.      $lowlink[v] \leftarrow \min\{lowlink[v], visitNum[w]\}$ ;
14.   if  $lowlink[v] = visitNum[v]$  then do {
15.     repeat
16.        $w \leftarrow \{pop\ vertex\ from\ T\}$ ;
17.     until  $w = v$ ;
18.   }
19. }
20. main program *****/
21.  $cnt \leftarrow 1$ ;  $T \leftarrow \emptyset$ ;
22. for all  $v \in V$  do  $visited[v] \leftarrow False$ ;
23. while  $w \in V$  and  $visited[w] = False$  do
24.   CONNECT( $w$ );

```

In Algorithm 1, the procedure *CONNECT*(v) perform the depth-first searches for sc-components.

A variable *cnt* is a count of the global visit order (line 2). At line 4, all the candidates for sc-components are stored in a first-in-last-out stack *T*. Line 8 updates *lowlink*(*v*) if a son of *v*, *w*, is found such that *lowlink*(*w*) < *lowlink*(*v*). Line 10 further updates *lowlink*(*v*) if the root of the sc-component containing *w* is an ancestor of *v* in the tree *T_i*. Line 11 identifies roots and those vertices above and including the root on the stack induce an sc-component. They are then popped out from the stack (Gibbons 1985, Tarjan 1972).

Algorithm 2 is an SSSP algorithm based on the sc-component decomposition. First of all, Algorithm 2 calls Algorithm 1 to compute sc-components V_r, V_{r-1}, \dots, V_1 in a topological order where V_1 is the first and V_r is the last sc-component computed. Then, the graph can be degenerated into an acyclic graph $\tilde{G} = (\tilde{V}, \tilde{E})$ where $\tilde{V} = \{V_r, V_{r-1}, \dots, V_1\}$ and $\tilde{E} = \{(v, w) \mid (v, w) \in E, v \in V_i, w \in V_j, 1 \leq i, j \leq r \text{ and } i \neq j\}$ (see Figure 2). In Algorithm 2, (V_i, V_j) indicates a pseudo-edge in degenerated graph \tilde{G} such that one end-point of the edge is in V_i and another end-point is in V_j . Variable $d[v]$ maintains a distance of a path to vertex *v*.

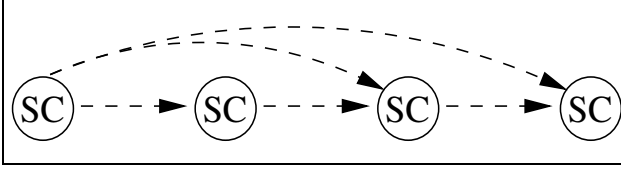


Figure 2: A degenerated acyclic graph \tilde{G} of the graph in Figure 1. \tilde{G} consists of sc-components and pseudo-edges connecting sc-components.

If we see each sc-component as a subgraph, then such a subgraph is defined as $G_i = (V_i, E_i)$ where $V_i \in \tilde{V}$ and $E_i = \{(v, w) \mid v \in V_i \text{ and } w \in V_i\}$. The SSSP problem from a source to all other vertices can be solved along the degenerated graph in the topological order from G_r to G_1 . For each subgraph G_i , we solve the *general shortest paths* (GSS) (Takaoka 1998) at line 5. The GSS is a generalized algorithm from Dijkstra's SSSP algorithm. In the *general shortest paths* algorithm, there is no source vertex superior to other vertices; all vertices have initial distances from the source *s*. The vertex with the minimum distance is chosen first, and corresponding updates are made, and so forth. The computation is similar to ordinary Dijkstra's SSSP algorithm except for the initial distance distribution. Readers are referred to (Takaoka 1998) for more details.

Algorithm 2. Solve the SSSP problem using sc-components decomposition

1. Compute sc-components V_r, V_{r-1}, \dots, V_1
2. **for** $v \in V$ **do** $d[v] \leftarrow \infty$;
3. $d[s] \leftarrow 0$; //For source *s* let $s \in V_r$ without loss of generality
4. **for** $i \leftarrow r$ **to** 1 **do** {
5. Solve the GSS for G_i ;
6. **for** V_j such that $(V_i, V_j) \in \tilde{E}$ **do**
7. **for** $v \in V_i$
8. **and** $w \in V_j$ such that $(v, w) \in E$ **do**
9. $d[w] \leftarrow \min\{d[w], d[v] + \text{cost}(v, w)\}$;
- }

Obviously, Algorithm 2 runs in $O(m)$ time.

Saunders and Takaoka (2005) offered an acyclic decomposition approach. In this approach, a graph

was decomposed into acyclic structures in $O(m)$ time. Each structure was dominated by a trigger vertex. We denote the 1-dominator set, which is the set of acyclic structures, by *D*. Note that triggers and acyclic structures correspond one to one. If a nearly acyclic graph had *r* trigger vertices, they introduced an algorithm with $O(m+r \log r)$ time complexity (Saunders & Takaoka 2005) for solving the SSSP problem. The new parameter *r* represented the number of acyclic structures in a graph. Intuitively speaking, an acyclic structure in a 1-dominator set is an acyclic subgraph such that any vertex inside can be reached from outside only through the associated trigger vertex. We say that the trigger dominates this acyclic structure. The 1-dominator set is the set of such acyclic structures. As the triggers and acyclic structures correspond one-to-one, we sometimes use the two concepts interchangeably. We also use “acyclic subgraph” and “acyclic structures” interchangeably. Take Figure 3 for example, there are three acyclic structures in the left picture.

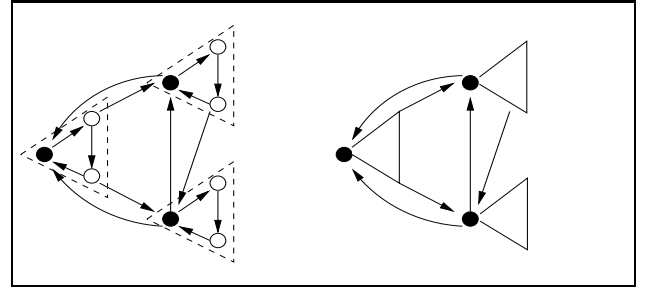


Figure 3: The acyclic structures in the left picture are presented in the right as combinations of a node and a triangle where the node represents a trigger vertex and the triangle represents non-trigger vertices dominated by the trigger vertex. There are three trigger vertices in the graph.

Algorithm 3 is a 1-dominator decomposition algorithm introduced by Saunders and Takaoka (2005) to implement the above statement.

In Algorithm 3, the procedure *rdfs*() stands for restricted depth-first search. It maintains variable *inCount* for each vertex *v*, which initially contains the total number of incoming edges of a vertex, $|IN(v)|$. When it *visits* a vertex *v* at lines 5-7, it decreases the *inCount* of the vertex by 1. If it is a new visit, then it will be added into a list *L* at line 6. If *inCount* becomes 0, we say it unlocks the vertex and the search goes forward (line 8), that is, the vertex is *traversed*. Unlocked vertices are included into a set *A* for the acyclic structure dominated by the initial vertex *w* (line 4). At the end of the search from *w*, *A* is the acyclic structure dominated by *w*, and *L* is the set of vertices visited (line 6). In this sense, we say, *L* is the set of visited vertices for possible inclusion into the acyclic structure. At the end $A[v]$ is the acyclic structure dominated by *v*, and $B[v]$ is the associated boundary set. The algorithm maintains $AC[v]$ which refers to an acyclic structure dominated by trigger vertex *v* (line 25), and $BS[v]$, which refers to the boundary set of the acyclic structure (line 26). The algorithm also maintains a queue *Q* containing boundary vertices as trigger vertex candidates (line 30). The algorithm starts from a vertex *s*, and searches with *rdfs*() as much as possible. Then it starts searches from boundary vertices again. We assume $n \leq m$ and only treat strongly connected graphs for simplicity. Generalization to a general graph is straightforward. As the graph is strongly connected, the search will eventually come back to *s*. We note that the searched

part from s is only traversed twice.

Algorithm 3. 1-dominator decomposition

```

1. function AcyclicSet( $w$ ) {
2.   VertexSet  $A, L, B$ ;
3.   procedure rdfs( $u$ ) {
4.      $A \leftarrow A + \{u\}$ ;
5.     for all  $v \in OUT(u)$  do {
6.       if  $v \notin L$  then  $L \leftarrow L + \{v\}$ ;
7.        $inCount[v] \leftarrow inCount[v] - 1$ ;
8.       if  $inCount[v] = 0$  //  $v$  unlocked
9.         then rdfs( $v$ );
10.    }
11.    $A \leftarrow \emptyset$ ;  $L \leftarrow \{w\}$ ;
12.    $inCount[w] \leftarrow inCount[w] + 1$ ;
13.   // prevents re-traversal of  $w$ 
14.   rdfs( $w$ );
15.   for all  $v \in L$  do  $inCount[v] \leftarrow |IN(v)|$ ;
16.   VertexSet  $B \leftarrow L - A$ ; // boundary vertices
17.   return ( $A, B$ );
18. }
19. /***** main program *****/
20. for all  $v \in V$  do  $vertexType[v] \leftarrow unknown$ ;
21. for all  $v \in V$  do  $inCount[v] \leftarrow |IN(v)|$ ;
22.  $Q \leftarrow \{s\}$ ;
23. while  $Q \neq \emptyset$  do {
24.   Remove the next vertex  $u$  from  $Q$ ;
25.   if  $vertexType[u] = unknown$  then {
26.     ( $A, B$ )  $\leftarrow$  AcyclicSet( $u$ );
27.     Let  $AC[u]$  refer to  $A$ ;
28.     Let  $BS[u]$  refer to  $B$ ;
29.      $vertexType[u] \leftarrow trigger$ ;
30.     for all  $v \in A$  do
31.        $vertexType[v] \leftarrow non-trigger$ ;
32.     for all  $v \in B$  do
33.       if  $vertexType[v] = unknown$  and  $v \notin Q$ 
34.         then Add  $v$  to  $Q$ ;
35.   }
36. }
```

The time complexity of Algorithm 3 is $O(m)$, which was proved by Saunders and Takaoka (2005).

Algorithm 4 is an SSSP algorithm based on the results of acyclic decompositions of graphs. It modifies a general SSSP algorithm introduced by Takaoka (1998). Obviously only trigger vertices will be added into the frontier set F (line 8). That means the delete-min operations will not be required by the non-trigger vertices. The distance values $d[v]$ of non-trigger vertices in an acyclic structure will be finalized straightaway with the *decreaseKey* operation in topological order after their associated trigger vertices reach the minimum values.

Algorithm 4. SSSP Algorithm Using Acyclic Decomposition

```

1. procedure decreaseKey( $u$ ) {
2.   for each  $v \in AC[u]$  in topological order do
3.     for each  $w \in OUT[v]$  and  $w \notin S$  do
4.        $d[w] = \min\{d[w], d[v] + cost(v, w)\}$ ;
5. }
6. /***** main program *****/
7. for all  $v \in V$  do  $d[v] = \infty$ ;
8. solution set  $S = \emptyset$ ;
9. insert all triggers into frontier set  $F$ ;
10. the source vertex  $s$  and  $d[s] \leftarrow 0$ ;
11. if  $s$  is not a trigger then decreaseKey( $s$ );
12. while  $F \neq \emptyset$  do {
13.    $d[u] = \min\{d[u] \mid u \in F\}$ ;
14.    $F \leftarrow F - u$ ; // delete-min
15.    $S \leftarrow S + u$ ;
16.   decreaseKey( $u$ );
17. }
```

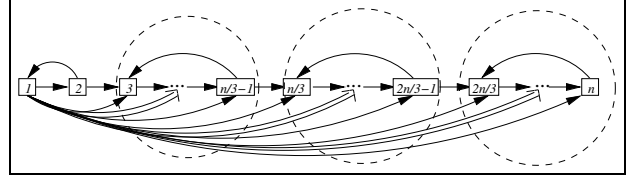


Figure 4: Arrow “ \Rightarrow ” indicates *Vertex 1* has edges to every node unstated. Let *Vertex 1* be the source, $cyc^1(G) = n/3$, $cyc^2(G) = 2$.

In line 8 of Algorithm 4, a trigger means a 1-dominator trigger. In Section 5, this will be a 1-2-dominator trigger.

3 Higher-Order Decomposition

In Takaoka’s approach, the sc-components originally computed in line 1 of Algorithm 2 remain fixed through all GSS’s. When we start one GSS, there is an opportunity that the corresponding sc-component can be further decomposed.

The basic idea behind *higher-order decomposition* is to remove the first vertex selected in GSS (line 5 of Algorithm 2), and then run the sc-component decomposition for the new strongly connected subgraphs. The original sc-component decomposition by Takaoka (1998) is called the *first order* decomposition, and the sc-component decomposition for the subgraphs is called the *second order* decomposition, one for sub-subgraphs is called the *third order* decomposition, and so forth.

After all distance updates are done in lines 6-8 of Algorithm 2, we solve the GSS for G_{i-1} . We call the first vertex whose distance is finalized the *pseudo source* of G_{i-1} .

When all the shortest paths have been computed, we find the largest sc-component in the decomposed graph. Let us indicate the size of the largest sc-component by ρ . An SSSP algorithm based on this approach has the worst case time complexity $O(hm + n \log \rho)$. It can be efficient for some nearly acyclic graphs like a graph in Figure 4.

Thus, we give another definition that the degree of cyclicity $cyc^h(G)$ is the maximum cardinality of the strongly connected components of the decomposed graph G after the h^{th} order decomposition is made.

Let $G = (V, E)$, $V = \{v_1, v_2, \dots, v_n\}$ and $E \subseteq V \times V$. We calculate sc-components V_r, V_{r-1}, \dots, V_1 . According to the new definition, we call the sc-components the *first order sc-components*, and they are defined as $V_r^1, V_{r-1}^1, \dots, V_1^1$. Then, we define the *first order degenerated graph* $\tilde{G}^1 = (\tilde{V}^1, \tilde{E}^1)$ where $\tilde{V}^1 = \{V_r^1, V_{r-1}^1, \dots, V_1^1\}$, $\tilde{E}^1 = \{(v \rightarrow w) \mid edge(v \rightarrow w) \subseteq E, v \in V_i^1 \text{ and } w \in V_j^1, i \neq j\}$ (see Figure 2).

We call edges in \tilde{E}_1 *transient edges* since they connect between sc-components. In contrast, we call an edge $(v \rightarrow w)$ an *inside edge* if both v and w belong to the same sc-component. Thus, graph \tilde{G}^1 is a decomposed acyclic graph of G [1]. Let us define each sc-components in $\tilde{G}^1 = (\tilde{V}^1, \tilde{E}^1)$ to be subgraphs. The subgraphs are defined as $G_i^1 = (V_i^1, E_i^1)$ where $V_i^1 \in \{V_r^1, V_{r-1}^1, \dots, V_1^1\}$, $E_i^1 = \{(v \rightarrow w) \mid v \in V_i^1 \text{ and } w \in V_i^1\}$ (see SC nodes in Figure 2).

We know that G_i^1 is strongly connected. When we erase all incoming edges of the *pseudo source vertex* of G_i^1 , immediately G_i^1 becomes not strongly connected. Based on this property, we use an *SDFS* algorithm (see Algorithm 5) to compute sc-

components $V_r^2, V_{r-1}^2, \dots, V_1^2$ for G_i^1 , and V_r^2 maintains the *pseudo source vertex*. Let us call sc-components $V_r^2, V_{r-1}^2, \dots, V_1^2$ of G_i^1 the *second order sc-components* of G_i^1 .

Algorithm 5 implements the idea of *higher-order decomposition* by modifying Algorithm 1. It is called *Selective Depth-First Search* (SDFS) algorithm. In this algorithm, a set SV maintains source or pseudo source vertices. In the following, SV is a singleton. This can be generalized into a set if we finalize distances for a few vertices in solving a GSS. We do not seek this possibility in Algorithm 6. Variable $Graph[w]$ maintains an index i of a subgraph \tilde{G}_i^h which w belongs to.

In Algorithm 5 at line 21 we initialize the vertices in SV to be *visited* because they are selected not to join the sc-components search. At line 22, we call the recursive procedure $CONNECT(v)$ until every vertex of the graph becomes *visited*. After all the sc-components of the graph or degenerated graph are determined, from line 23 to line 26, we save vertex $v \in SV$ into a separate sc-component.

Now we look at the recursive procedure $CONNECT(v)$ (line 1). At line 8, if w is already in a sc-component, we reserve this edge in \tilde{E}^h , and erase the edge from $OUT(v)$. The reason of erasing the edge from $OUT(v)$ is that the edge connects strongly connected subgraphs, and we do not visit the edge again if we have to further decompose the subgraph which v belongs to. When $CONNECT(v)$ has processed all the outgoing edges of v , $lowlink[v] = visitNum[v]$ if and only if v is the root of the sc-component containing v (line 10). If line 10 is satisfied, we determine an sc-component (line 11 to 15).

Algorithm 5. Selective Depth-First Search algorithm (SDFS). In the first order decomposition, $SV = \{s\}, h = 1$.

```

1. procedure CONNECT( $v$ )
2. {
3.    $visited[v] \leftarrow True$ ;  $visitNum[v] \leftarrow cnt$ ;
    $cnt \leftarrow cnt + 1$ ;  $lowlink[v] \leftarrow visitNum[v]$ ;
    $Graph[v] \leftarrow \infty$ ; //  $\infty > n$ 
4.    $T \leftarrow T + v$ ;
5.   for each  $w \in OUT(v)$  {
6.     if  $visited[w] = False$  then {
        $CONNECT(w)$ ;
        $lowlink[v] \leftarrow \min\{lowlink[v], lowlink[w]\}$ ;
     }
7.     else if  $visitNum[w] < visitNum[v]$ 
       and  $w \in T$  then
        $lowlink[v] \leftarrow \min\{lowlink[v], visitNum[w]\}$ ;
8.     if  $1 \leq Graph[w] < r$  then {
        $\tilde{E}_i^h \leftarrow \tilde{E}_i^h + (v \rightarrow w)$ ; // transient edge
        $OUT(v) \leftarrow OUT(v) - (v \rightarrow w)$ ;
       //  $OUT(v)$  is a set of inside edges at the end
     }
9.   }
10.  if  $lowlink[v] = visitNum[v]$  then do {
11.    repeat
12.       $w \leftarrow popvertexfromT$ ;
       $V_r \leftarrow V_r + w$ ;  $Graph[w] \leftarrow r$ ;
13.    until  $w = v$ ;
14.     $r \leftarrow r + 1$ ;
15.  }
16. }
17. function SDFS( $G_i^h, SV$ ) // graph  $G_i^h$  to be
    decomposed with source in set  $SV$ .
18. {
19.    $r \leftarrow 1$ ;  $cnt \leftarrow 1$ ;  $T \leftarrow \emptyset$ ;
20.   for each  $v \in V$  do {

```

```

    $visited[v] \leftarrow False$ ;
    $Graph[v] \leftarrow NULL$ ;
21. }
22. for  $v \in SV$  do  $visited[v] \leftarrow True$ ;
23. while  $w \in V$  and  $visited[w] = False$  do
    $CONNECT(w)$ ;
24. for  $v \in SV$  do {
25.    $V_r \leftarrow V_r + v$ ;
26.   for  $w \in OUT(v)$  do
     if  $1 \leq Graph[w] < r$ 
     then  $\tilde{E}_i^h \leftarrow \tilde{E}_i^h + (v \rightarrow w)$ ;
27. }

```

Algorithm 6 is an SSSP algorithm based on the higher order decomposition. It is generalized from Algorithm 2. The algorithm starts a recursive call a procedure $Dynamic(G, SV)$ at line 16, and $SV = \{s\}$ because it always starts from the first order decomposition, so the source vertex is the only vertex in set SV . Now let us look at the procedure $Dynamic(G, SV)$. At line 2 we call Algorithm 5 to compute the h^{th} order sc-components V_r^h, \dots, V_1^h . Note that the r will be different from one decomposition to another. From line 4 to line 6, we update $d[w]$ according to edges $(v \rightarrow w)$, $v \in V_i^h$ and $w \in V_j^h$, ($r + 1 \geq i > j$). At line 7 we find its *pseudo source vertex* for a h^{th} order subgraph G_i^h , and then update the set SV of it (line 8). At line 9, we terminate the recursive call if conditions ($|G_i^h| - 1 > c_1$) or ($h + 1 \leq c_2$) are satisfied, c_1, c_2 are constants. The program will decompose the graph until each V_i^h has only c_1 vertices or the graph has been decomposed c_2 times. Generally speaking, the bigger c_2 is, the more sc-components we will have. After terminating the recursive call, line 10 solves the SSSP problem for the subgraph G_i^h .

Algorithm 6. For the second order decomposition, $c_2 = 2$.

```

1. procedure Dynamic( $G, SV$ ) {
2.   Call SDFS( $G, SV$ ) to compute  $h^{th}$  order
   sc-components  $V_r^h, \dots, V_1^h$ ;
3.   for  $i \leftarrow r$  to 1 do {
4.     for  $V_j^h$  such that  $(V_i^h, V_j^h) \in \tilde{E}^h$  do
5.       for  $v \in V_i^h$  do for  $w \in V_j^h$  do
6.          $d[w] \leftarrow \min\{d[w], d[v] + cost(v, w)\}$ ;
7.        $v_{min} \leftarrow w$  that gives
          $\min\{d[w] \mid w \in V_{i-1}^h\}$ ;
8.      $SV \leftarrow \{v_{min}\}$ ;
9.     if ( $|G_i^h| > c_1$ ) and ( $h + 1 \leq c_2$ )
       then do {
          $h \leftarrow h + 1$ ;
          $Dynamic(G_i^h, SV)$ ;
       }
10.    else Solve the SSSPs for  $G_i^h$ ;
11.  }
12. }
13. /***** main program *****/
14. for  $v \in V$  do  $d[v] \leftarrow \infty$ ;
15. the source vertex  $s$  and  $d[s] \leftarrow 0$ ;
16.  $h \leftarrow 1$ ;
17.  $Dynamic(G, \{s\})$ ;

```

4 A Combined Measure of Acyclicity

In this section we combine the two measures of near acyclicity (see Figure 5). Conceptually we first obtain the 1-dominator set, and degenerate the graph G to

G' where the vertices of G' are the triggers and an edge from u to v in G' exists if an edge exists from some vertex in $AC[u]$ to v . We sometimes refer to this edge in G' as a pseudo edge. Then we search for sc-components in the degenerated graph G' . Obviously the maximum size of the sc-components in the above is good enough for the size of the priority queue in the SSSP algorithm.

In the real programming, Algorithm 7 calls Algorithm 3 for triggers as a subroutine. Specifically it calls *AcyclicSet* at line 3, and only puts the trigger vertex v into a first-in-last-out stack T (line 11). If v can reach another trigger vertex w with a lower visit number, $visitNum[w]$, then v can not be the root of an sc-component in the depth-first search tree. We record this reachability in an array $lowlink[]$. When the depth-first-search finishes, the trigger vertices recorded in T will be popped out until a root vertex u if $visitNum[u] = lowlink[u]$, to form an sc-component (line 16). See Figure 5 for an example, where the degree of cyclicity of the graph G' is 3.

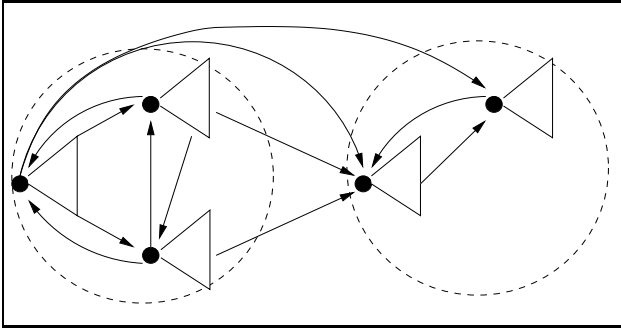


Figure 5: A graph with combined nearly acyclic structures.

In order to implement this approach, the visit number $visitNum$ and the low-link number, $lowlink$, of triggers must be computed,

In Algorithm 7, global variables c and p refer to how many vertices are visited and how many sc-components are identified respectively. Other variables are explained in Table 1.

We call Algorithm 7 the hierarchical depth first search algorithm (HDFS). It first calls function *AcyclicSet*(v) to identify a vertex set $AC[v]$ of an acyclic structure with associated trigger vertex v and boundary vertices of $AC[v]$ (line 3). All the acyclic subgraphs are computed through recursive calls of *hdfs* (line 12), and we can identify the number of triggers, $r = |D|$. In the mean time, the values of $lowlink[v]$, for dominating vertex v of every acyclic part, will be updated from the boundary vertices (lines 13 and 14). If any u of the boundary vertices is *unvisited* ($visitNum[u] = 0$), the depth first search will carry on from that vertex until all boundary vertices update their $lowlink[v]$ values. When a depth first search completes, it checks whether a trigger is a root of an sc-component (line 16). If it is a root, an sc-component of triggers is detected, and we can have the number of triggers in an sc-component, say l_i . Eventually, let $l = \text{Max}\{l_1, l_2, \dots, l_p\}$, then we will have $l = \text{cyc}(G')$ where $\text{cyc}(G')$ is the degree of cyclicity of the degenerated graph G' . The time for Algorithm 7 is $O(m)$, as each edge is examined only once.

Algorithm 7. Hierarchical Depth First Search

1. **function** *HierarchySets*(v_0) {
2. **procedure** *hdfs*(v) {
3. $(A, B) \leftarrow \text{AcyclicSet}(v)$;
4. Let $AC[v]$ refer to A ;

Table 1: Notations used in algorithms and proofs (sort alphabetically)

A	an acyclic part computed by function <i>AcyclicSet</i> ()
$AC[v]$	reference value pointing to an acyclic structure dominated by vertex v
B	a boundary vertex set
$BS[v]$	reference value pointing to a boundary vertex set of an acyclic structure $AC[v]$
C	an sc-component computed by function <i>HierarchySets</i> ()
c	count of the number of visited vertices
$\text{cost}(v, w)$	cost of edge $(v \rightarrow w)$
$d[v]$	distance of a path to vertex v
$IN(v)$	$\{v \mid (w \rightarrow v) \in E\}$
$\text{inCount}[v]$	number of untraversed incoming edges of vertex v
L	a vertex set maintains all vertices visited in function <i>AcyclicSet</i>
$lowlink[v]$	pointing to the root of an sc-component which v belongs to.
$OUT(w)$	$\{w \mid (w \rightarrow v) \in E\}$
p	count of the number of sc-components
$SC[p]$	reference value pointing to an sc-component sorted at topological order p
T	first-in-last-out stack
$visitNum[v]$	visited order of vertex v

```

5.   vertexType[v] ← trigger;
6.   for all  $u \in A$  do
7.     vertexType[u] ← non-trigger;
8.   for all  $u \in B$  do
9.     if  $visitNum[u] = 0$  do {
10.       $c \leftarrow c + 1$ ;  $visitNum[u] \leftarrow c$ ;
11.       $lowlink[u] \leftarrow c$ ;
12.       $T \leftarrow T + \{u\}$ ;
13.      hdfs( $u$ ); // search from unvisited  $v \in B$ 
14.      if  $u \in T$  then
15.         $lowlink[v] \leftarrow \text{Min}(lowlink[u], lowlink[v])$ ;
16.      else do
17.         $lowlink[v] \leftarrow \text{Min}(lowlink[v], visitNum[u])$ ;
18.      // update  $lowlink[v]$  from connected triggers
19.    }
20.  if  $lowlink[v] = visitNum[v]$  and  $v \in T$  do {
21.    VertexSet  $C \leftarrow \{\text{pop up vertices from } T \text{ until } v\}$ ;
22.     $p \leftarrow p + 1$ ; // count sc-components
23.    Let  $SC[p]$  refer to  $C$ ;
24.  }
25. }
26. /***** main program *****/
27. VertexSet  $AC \leftarrow \emptyset$ ,  $SC \leftarrow \emptyset$ ,  $T \leftarrow \emptyset$ ;
28. for all  $v \in V$  do {
29.    $\text{inCount}[v] \leftarrow |IN(v)|$ ;
30.    $\text{vertexType}[v] \leftarrow \text{unknown}$ ;
31.    $visitNum[v] \leftarrow 0$ ;  $lowlink[v] \leftarrow \infty$ ; }
32.  $p \leftarrow 0$ ;  $c \leftarrow 0$ ;
33. for all unvisited  $v_0 \in V$  do {
34.    $c \leftarrow c + 1$ ;  $visitNum[v_0] \leftarrow c$ ;  $lowlink[v_0] \leftarrow c$ ;
35.    $T \leftarrow T + \{v_0\}$ ;
36.   hdfs( $v_0$ );
37. }
38. return ( $AC$ ,  $SC$ ,  $p$ );
39. }
```


Algorithm 8 modifies Algorithm 4. It is an SSSP algorithm using topologically sorted sc-components and topologically sorted vertices in each acyclic structure computed by Algorithm 3. Obviously only trigger vertices in an sc-component will be added into the frontier set F in a topological order (lines 11-14). That means the non-trigger vertices will not do the delete-min operations. The distance values $d[v]$ of the non-trigger vertices in an acyclic subgraph will be decreased straightaway when their associated trigger vertices reach the minimum values.

In Algorithm 8, we assume that sc-components are topologically sorted in the order $p, \dots, 1$. That is, if there is an edge $(v \rightarrow w)$, $v \in SC[i]$, $w \in SC[j]$, then $i > j$. Let us assume that an acyclic component A has k vertices, and these vertices are topologically sorted in the order v_1, \dots, v_k . That is, $\forall 1 \leq i, j \leq k$ if $(v_i, v_j) \in E \Leftrightarrow i < j$. Here, vertex v_1 is the trigger vertex of A .

Algorithm 8. SSSP Algorithm Using Hierarchical Decomposition

```

1. procedure decreaseKey( $u$ ) {
2.   for each  $v \in AC[u]$  in topological order do
3.     for each  $w \in OUT(v)$  and  $w \notin S$  do
4.        $d[w] = \text{Min}\{d[w], d[v] + \text{cost}(v, w)\}$ ;
5. }
/***** main program *****/
6. for all  $v \in V$  do  $d[v] \leftarrow \infty$ ;
7. the source vertex  $s$  and  $d[s] \leftarrow 0$ ;
8. solution vertex set  $S \leftarrow \emptyset$ ;
9.  $(AC, SC, p) \leftarrow \text{HierarchySets}(s)$ ;
10. if  $s$  is not a trigger then decreaseKey( $s$ );
11. for  $i \leftarrow p$  to 1 do {
12.   front vertex set  $F \leftarrow \emptyset$ ;
13.   for  $v \in SC[i]$  do
14.     if  $v$  is a trigger then insert  $v$  into  $F$ ;
15.   while  $F \neq \emptyset$  do {
16.      $d[u] = \text{Min}\{d[u] \mid u \in F\}$ ;
17.      $F \leftarrow F - u$ ; // delete-min
18.      $S \leftarrow S + \{u\}$ ;
19.     decreaseKey( $u$ );
20.   }
21. }
```

The computing time of Algorithm 8 is $O(m + \sum_{i=1}^p l_i \log l_i)$ with condition that $l_i \leq l$ and $\sum_{i=1}^p l_i = r$. This time is maximized when we set as many l_i 's to l as possible, and we have the time is bounded by $O(m + r \log l)$.

5 1-2-dominator Set

A 1-2-dominator set is similar to a 1-dominator set. It also decomposes a graph into a collection of acyclic subgraphs, where each subgraph is dominated by one vertex only, or two trigger vertices cooperatively. It offers potentially larger acyclic subgraphs than a 1-dominator set does. Thereby, fewer trigger vertices are needed to cover the whole graph, that is, $r' \leq r$, where r' is the number of triggers in a 1-2-dominator decomposition, and r is the number of triggers in a 1-dominator set. Considering efficient shortest path algorithms only do delete-min operations on trigger vertices, fewer trigger vertices can reduce the time for solving the SSSP problem. When r' is much smaller than r , we can efficiently compute SSSPs in $O(m + r' \log r')$ time.

We can run Algorithm 8 using the results of 1-2-dominator decompositions. In the worst case, two trigger vertices u and v may cooperatively dominate exactly the same acyclic part of the 1-dominator decomposition, that is, $AC[u] = AC[v]$. Then, in the

worst case, a single-source computation using a 1-2-dominator set will scan each acyclic part up to two times. The corresponding worst-case time complexity of the algorithm is $O(2m + r' \log r') = O(m + r' \log r')$, where constant 2 is emphasized in the left-hand side.

Now, we shall show how to compute a 1-2-dominator set. First we compute a 1-dominator set of a given graph. In order to save the time for computing a 1-2-dominator set, we degenerate each acyclic structure into its associated trigger vertex during the 1-dominator decomposition. In Figure 6 the upper picture is degenerated into the lower picture with pseudo edges represented by " \Rightarrow ". We assign a boundary vertex set $BS[v]$ for the trigger vertices. For a boundary vertex w in a set $BS[v]$, we save the number of outgoing edges from $AC[v]$ to w , $inc(v, w)$, together with w in BS .

Algorithm 9 implements this process. It modifies the function *AcyclicSet* of Algorithm 3 with line numbers extended with dots (lines 15-15.4). The main program of Algorithm 3 is the same. In this algorithm, a vertex set A reserves all the non-trigger vertices dominated by the associated trigger vertex, a vertex set L reserves all vertices visited by the current search, a vertex set B contains all the boundary vertices of an acyclic structure.

Algorithm 9. Modified Function for Computing AC and BS

```

1. function AcyclicSet( $w$ ) {
2.   VertexSet  $A, L, B$ ;
   //  $B$  is enhanced with  $inc$ 
3.   procedure rdfs( $u$ ) {
4.      $A \leftarrow A + u$ ;
5.     for all  $v \in OUT(u)$  do {
6.       if  $v \notin L$  then  $L \leftarrow L + \{v\}$ ;
7.        $incCount[v] \leftarrow incCount[v] - 1$ ;
8.       if  $incCount[v] = 0$  then rdfs( $v$ );
9.     }
10.  }
11.   $A \leftarrow \emptyset$ ;  $L \leftarrow \{w\}$ ;
12.   $incCount[w] \leftarrow incCount[w] + 1$ ;
13.  rdfs( $w$ );
14.  VertexSet  $B \leftarrow L - A$ ; // boundary vertices
15.  for each  $v \in B$  do {
15.1     $inc(v_0, v) = |IN(v)| - incCount[v]$ ;
15.2     $BS[v_0] \leftarrow (v, inc(v_0, v))$ ;
15.3     $incCount[v] \leftarrow |IN(v)|$ ;
15.4  }
16.  return ( $A, B$ );
17. }
/***** main program *****/
18. for all  $v \in V$  do vertexType[ $v$ ]  $\leftarrow$  unknown;
19. for all  $v \in V$  do  $incCount[v] \leftarrow |IN(v)|$ ;
20.  $Q \leftarrow \{s\}$ ;
21. while  $Q \neq \emptyset$  do {
22.   Remove the next vertex  $u$  from  $Q$ ;
23.   if vertexType[ $u$ ] = unknown then {
24.      $(A, B) \leftarrow \text{AcyclicSet}(u)$ ;
25.     Let  $AC[u]$  refer to  $A$ ;
26.     vertexType[ $u$ ]  $\leftarrow$  trigger;
27.     for all  $v \in A$  do
28.       vertexType[ $v$ ]  $\leftarrow$  non-trigger;
29.     for all  $v \in B$  do
30.       if vertexType[ $v$ ] = unknown and  $v \notin Q$ 
31.         then Add  $v$  to  $Q$ ;
32.   }
```

After we finish the computation of 1-dominator decomposition, we look up the tables of boundary sets $BS[v]$, for a 1-dominator trigger v , to do the 1-2-dominator decomposition, rather than traverse the whole graph again.

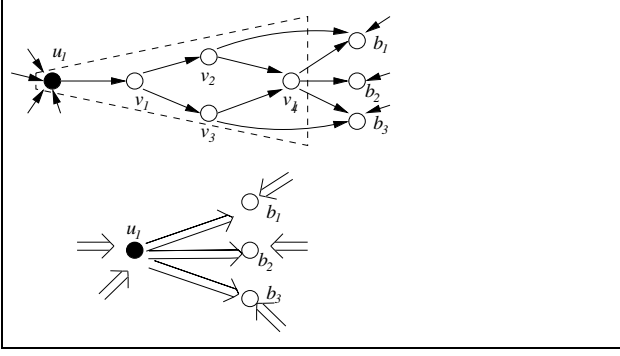


Figure 6: Simplified acyclic structure. Arrows “ \Rightarrow ” represent many edges to or from the same trigger vertex.

From one 1-dominator trigger, say u_1 , we look for a partner, say u_2 , which can co-dominate some part of the graph. The co-dominated part is given to both $AC[u_1]$ and $AC[u_2]$. Unlocked vertices are excluded from the pool of triggers (line 5). The generalized depth-first search $gdfs()$ goes over the degenerated graph.

Algorithm 10 implements this design. In Algorithm 10, a vertex set T_1 reserves all the 1-dominator trigger vertices. Note that $BS[v]$ in function $gdfs()$ plays the role of $OUT(v)$ in Algorithm 3.

Algorithm 10. 1-2-dominator Set Algorithm

```

1. procedure  $gdfs(u_1, u_2, v)$  {
2.   for all  $w \in BS[v]$  {
3.      $inCount[w] \leftarrow inCount[w] - inc(v, w)$ ;
4.     if  $inCount[w] = 0$  do { //  $w$  is unlocked
5.        $T_1 \leftarrow T_1 - \{w\}$ ;
6.        $AC[u_1] \leftarrow AC[u_1] + \{w\}$ ;
7.        $AC[u_2] \leftarrow AC[u_2] + \{w\}$ ;
8.        $gdfs(u_1, u_2, w)$ ;
9.     }
10.  }
11. }
12. /***** main program *****/
13. Compute 1-dominator set  $T_1$  by Algorithm 9;
14. for  $u_1 \in T_1$  do {
15.    $inCount[u_1] \leftarrow inCount[u_1] + 1$ ;
16.   for all  $v \in BS[u_1]$  do
17.      $inCount[v] \leftarrow inCount[v] - inc(u_1, v)$ ;
18.     for  $u_2 \in T_1 - \{u_1\}$  do {
19.        $inCount[u_2] \leftarrow inCount[u_2] + 1$ ;
20.        $gdfs(u_1, u_2, u_2)$ ;
21.     }
22.    $T_1 \leftarrow T_1 - \{u_1\}$ ;
23.   for all  $v \in BS[u_1]$  do  $inCount[v] \leftarrow |IN(v)|$ ;
24. }
```

At line 12, it takes $O(m)$ to complete a 1-dominator set computation. From line 13 to 23, it takes $O(r^2)$ to check all the possible pairs of 1-dominator triggers. The total time spent by $gdfs$ is $O(m')$ where m' is the number of pseudo edges bounded by m . So, the time complexity of Algorithm 10 is $O(m + r^2)$. Algorithm 9 decomposes set V into 1-dominator structures. Algorithm 10 in fact decomposes V into a mixture of 1-dominator structures and 1-2-dominator ones. This decomposition is set wise unique as the 1-dominator decomposition. More rigorous proof for uniqueness will be given in a future report.

To solve the SSSP problem, we can use Algorithm 8 with the set of remaining triggers computed by Algorithm 10. Algorithm 8 will perform *decreaseKey*

operations on 1-dominator structures once each, and twice on 1-2-dominator structures.

Once the 1-2-dominator set has been computed, we can repeatedly use it for computing SSSPs. When r' is much smaller than r , the decomposition time of the 1-2-dominator set will be balanced off by the time saved from repeatedly computing SSSPs, when only edge costs change in the same graph structure.

6 Evaluation

The new shortest path algorithms presented in this paper are theoretically more efficient than other algorithms for solving the SSSP problem on suitable nearly acyclic graphs. This offers a potential improvement on the running time in practice. In this section, we will look at what exact improvement is possible.

Experimented graphs are line-spanning random graphs. That is, for a graph $G = (V, E)$ with n vertices in V and initially no edges in E , we first add all the edges (v_i, v_{i+1}) for $1 \leq i < n$ into E . Then we repeatedly generate edge $(v \rightarrow w)$ at random such that $v \in V, w \in V, (v \rightarrow w) \notin E$ and $v \neq w$. Here, we use an edge factor f to specify those randomly added edges. Therefore, the total number of edges $m = (1 + f)n$ (Saunders 2004). Each edge has a cost between 1 and 100.

For algorithms presented in sections 4 and 5, we run them in graphs with different nearly acyclic structures. One kind of graph has nearly acyclic structures where the degree of cyclicity is $cyc(G)$ small (see Figure 7 for an example). Another kind of graph has few simple cycles for the graph size (see Figure 8 for an example). The third kind of graph has a combined structures (see Figure 9 for an example). In those experiments, graphs have n vertices and $2.8n$ edges. The number of vertices in the graphs start at 2,000 for Figures 7 and 8 and 2,197 for Figure 9 and doubling for successive values of n up until 128,000 for Figures 7 and 8 and 79,507 for Figure 9. This provides a large enough window to demonstrate the overall trends in algorithm performance.

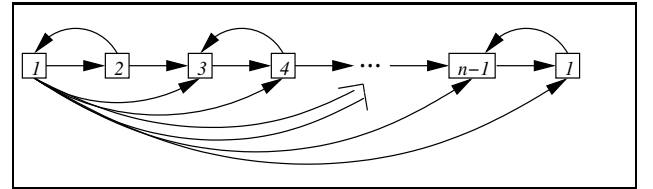


Figure 7: Arrow “ \Rightarrow ” indicates Vertex 1 has edges to each other node $i, 4 < i < n - 1$, in the graph. Let Vertex 1 be the source, $cyc(G) = 2, r = n/2, l = 2$

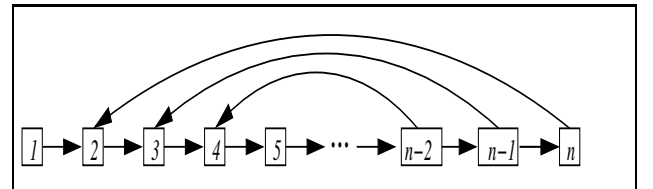


Figure 8: Let Vertex 1 be the source, $cyc(G) = n - 1, r = 3, l = 3$.

Three algorithms have been implemented (see Table 2) for solving the SSSP problem, and the SSSP computation time has been measured for analysis. Figure 10 shows that the new hierarchical approach performs as efficiently as that of the *SC* approach in graphs of Figure 7. We call such graphs *SC* approach favored graphs. Figure 11 shows that the

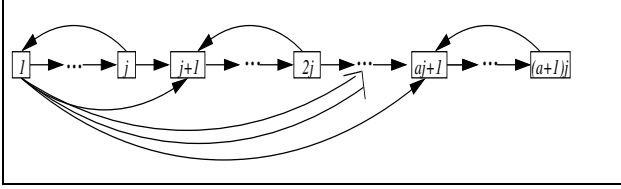


Figure 9: $j = \sqrt{n}$ and $a = \sqrt{n}-1$. Arrow “ \Rightarrow ” indicates *Vertex* 1 has edges to every node $ij + 1$, $1 < i < a$, in the graph. Let *Vertex* 1 be the source, $cyc(G) = \sqrt{n}$, $r = \sqrt{n}$, $l = 1$.

Table 2: The different algorithms implemented in experiments.

Name	Description
Acyclic	Acyclic approach, Saunders & Takaoka (2005)
Hierarchical	The new hierarchical approach
SC	Strongly connected approach, Takaoka (1998)

new approach performs the SSSP computation as efficiently that of the acyclic approach in acyclic approach favored graphs of Figure 8. Figure 12 shows that the new approach can outperform the other two approaches in graphs with combined nearly acyclic structures (see the graph in Figure 9).

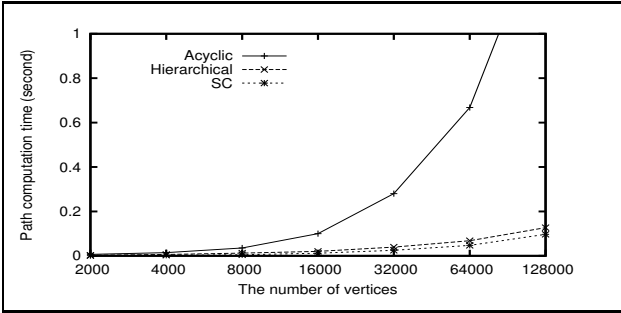


Figure 10: Evaluation of algorithms solving SSSP problem for graphs presented in Figure 7.

For algorithms presented in Section 5, in Figure 13 there are proportions of triggers in 1-dominator sets and 1-2-dominator sets. In the experiments, there are 1,000 vertices in each graph, but the number of edges varies from $1.1 \times 1,000$ to $13.8 \times 1,000$. The abscissa shows the edge factor of each graph. From the Figure 13, we can see that the number of triggers in 1-2-dominator sets is about 20 percent smaller than that in the 1-dominator sets in sparse graphs with edge number between $1.2n$ and $4.2n$.

In Figure 14, there is the time for solving the SSSP problem between using 1-dominator sets and using 1-2-dominator sets. Graphs have 1,000 vertices and edge factors varies from 0.1 to 0.25. The ordinate shows the computation time measured in seconds. We can see from this figure that when the graph is sparse and the edge factor is between 0.005 and 0.25, it reflects the efficiency gained from the reduced number of triggers in 1-2-dominator sets. When the graph becomes denser and the edge factor gets closer to 0.25, the difference of computation time gets smaller and smaller until there is almost no difference at 0.25.

7 Concluding Remarks

In this paper, we introduce three new approaches and algorithms based on them to solve the single-

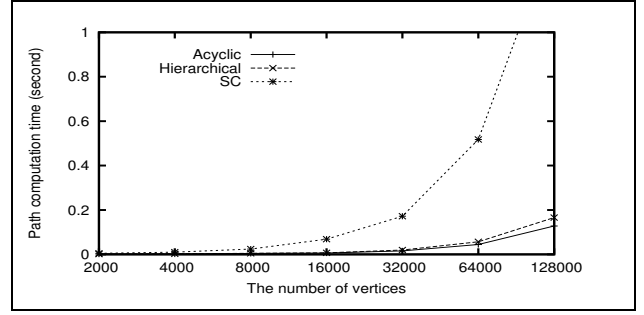


Figure 11: Evaluation of algorithms solving SSSP problem for graphs presented in Figure 8.

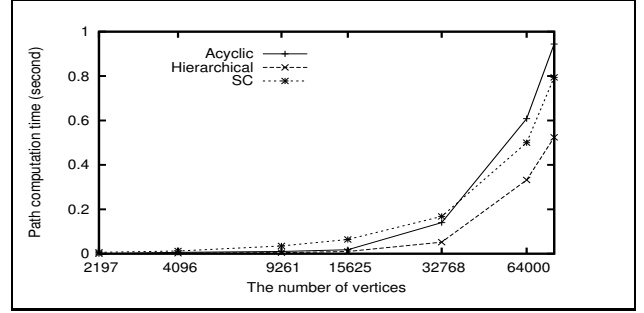


Figure 12: Evaluation of algorithms solving SSSP problem for graphs presented in Figure 9.

source shortest path (SSSP) problem in nearly acyclic graphs. We first present a *higher-order* approach to decompose a graph into strongly connected components (sc-components) as small as possible. Under *higher-order decomposition* framework, we give another definition of acyclicity, which is generalized from the definition of Takaoka’s (1998). That is, the degree of cyclicity is the maximal cardinality of the strongly connected components of the decomposed graph G after the h^{th} order decomposition is made, and the degree of cyclicity is denoted by $cyc^h(G)$. When a graph is decomposed into sc-components, we run Dijkstra’s algorithm only for each sc-component but not the whole graph. When all sc-components are small and the degree of acyclicity $cyc^h(G) = \rho$, we can efficiently compute SSSPs in $O(hm + n \log \rho)$ time. If we repeatedly use the decomposed graph for solving the SSSP problem, we can balance the time used for preprocessing the graph.

Then, we show that we can compute acyclic components and sc-components in a graph at the same time. The benefit of merging these two graph preprocessing approaches is to make a superior measure over the two existing measures of nearly acyclic graphs. Let r be the number of trigger vertices in a

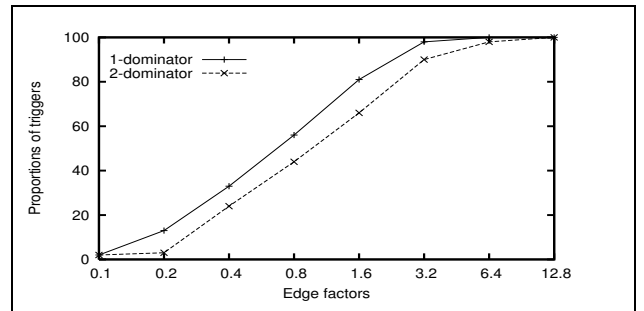


Figure 13: The proportion of triggers in 1-dominator sets and 1-2-dominator sets.

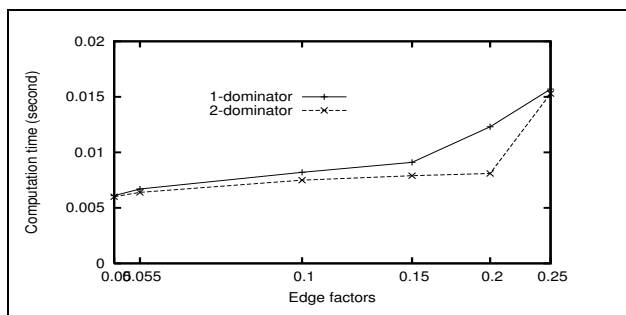


Figure 14: the computing time for SSSP problem between using 1-dominator sets and using 1-2-dominator sets in different density graphs.

1-dominator set, and $l = cyc(G')$ where $cyc(G')$ is the degree of the acyclicity of the degenerated graph G' . When l or r is small, we say that the graph is nearly acyclic, and we can efficiently solve the SSSP problem for the graph in $O(m+r\log l)$ time, which takes advantage of both measures.

We also present a demonstration of a 1-2-dominator set, which can be generalized to a multidominator set, denoted by a k -dominator set. In the 1-2-dominator set, one or two trigger vertices cooperatively dominate an acyclic structure in a graph. This offers potentially larger acyclic structures than the 1-dominator set does. Thereby, fewer trigger vertices are needed to cover the whole graph, that is, $r' \leq r$, where r' is the number of triggers in the 1-2-dominator decomposition, and r is the number of trigger vertices in the 1-dominator set. Considering efficient shortest path algorithms only do delete-min operations on trigger vertices, fewer trigger vertices can reduce the time for computing shortest paths. Once the 1-2-dominator set has been computed, we can repeatedly use it for computing SSSPs. When r' is much smaller than r , the decomposition time of the 1-2-dominator set will be balanced off by the time saved from repeatedly computing SSSPs. It is open whether we can compute the 1-2-dominator set in $O(m)$ time.

References

- Abuaiadh, D. & Kingston, J. H. (1993), 'An Efficient Algorithm for the Shortest Path Problem', *Technical Report* **473**.
- Abuaiadh, D. & Kingston, J. H. (1994), 'Efficient Shortest Path Algorithms By Graph Decomposition', *Technical Report* **475**.
- Dijkstra, E. W. (1959), 'A note on two problems in connexion with graphs', *Numerische Mathematik* **1**, 269–271.
- Frederickson, G. N. (1987), 'Fast algorithm for shortest path in planar graph with applications', *SIAM Journal on Computing* **16**, 1004–1022.
- Fredman, M. L. & Tarjan, R. E. (1987), 'Fibonacci heaps and their uses in improved network optimization algorithms', *Journal of the ACM* **34**, 569–615.
- Alan Gibbons. (2004), *Algorithm Graph Theory*, Cambridge University Press.
- Saunders, S. (2004), Improved Shortest Path Algorithms for Nearly Acyclic Graphs, PhD thesis, Department of Computer Science and Software Engineering, University of Canterbury, New Zealand.
- Saunders, S. & Takaoka, T. (2003), 'Improved Shortest Path Algorithms for Nearly Acyclic Graphs', *Theoretical Computer Science* **293**(3), 535–556.
- Saunders, S. & Takaoka, T. (2005), 'Efficient Algorithm for Solving Shortest Paths on Nearly Acyclic Directed Graphs', *CATS* **41**.
- Takaoka, T. (1998), 'Shortest Path Algorithm for Nearly Acyclic Directed Graphs', *Theoretical Computer Science* **203**(1), 145–150.
- Takaoka, T. (2003), 'Theory of 2-3 Heaps', *Discrete Applied Mathematics* **126**, 115–128.
- Tarjan, R. (1972), 'Depth-first search and linear graph algorithms', *SIAM Journal on Computing* **1**(2), 146–160.
- Tarjan, R. (1983), 'Data Structures and Network Algorithms', *Society for Industrial and Applied Mathematics* **44**.

From Graphs to Euclidean Virtual Worlds: Visualization of 3D Electronic Institutions

Sara Drago¹Anton Bogdanovych²
Helmut Berger³Massimo Ancona¹
Carles Sierra⁴Simeon Simoff²

¹Department of Computer Science, University of Genoa,
via Dodecaneso 35, 16146 Genova, Italy
{drago,ancona}@disi.unige.it

²Faculty of Information Technology, University of Technology Sydney,
Sydney, NSW, Australia
{anton,simeon}@it.uts.edu.au

³Electronic Commerce Competence Center - EC3
Donau-City-Strasse 1, A-1220 Wien, Austria
helmut.berger@ec3.at

⁴Artificial Intelligence Research Institute (IIIA-CSIC)
Barcelona, Catalonia, Spain
sierra@iiia.csic.es

Abstract

In this paper we propose an algorithm for automatic transformation of a graph into a 3D Virtual World and its Euclidean map, using the rectangular dualization technique. The nodes of the initial graph are transformed into rooms, the connecting arcs between nodes determine which rooms have to be placed next to each other and define the positions of the doors connecting those rooms. The proposed algorithm is general enough to be used for automatic generation of 3D Virtual Worlds representation of any planar graph, however, our research is particularly focused on the automatic generation of 3D Electronic Institutions from the Performative Structure graph.

Keywords: 3D Electronic Institutions, Rectangular Dualization, Virtual Worlds.

1 Introduction

Nowadays, our society becomes extremely experience focused. The software developers and designers take significant care to providing good user experience in their products. In other industries like entertainment, business, travel and even health care experiences are bought and sold together with products and services or even separately from them. One of the paradoxes of this new economical phenomenon is that those experiences sometimes become much more valuable (expensive) than the products or services they adhere to (Pine & Gilmore 1998).

The advent of communication networks and increasing computational power of the hardware created new possibilities of enhancing the user experience in software products, helping to adapt to the new economical circumstances. One of the quite recent technologies, 3D Virtual Worlds, provides the exciting possibility to create systems that look similar to our everyday life visual surroundings. Due to

this similarity, such systems literally help to immerse users into software products. With 3D modelling we can imitate real objects and profit from users' familiarity with their affordances¹. The use of avatars² implicitly incorporates location awareness and offers mechanisms for social interaction, supporting to a certain extent the way humans operate and interact in the real world. In this way social interaction between participants in an immersive environments becomes an important integral feature of software solutions.

3D Electronic Institutions are a step towards such new methods of software design for the *open* systems that are based on the metaphor of 3D Virtual Worlds. The unique characteristic of open systems is that their components are unknown beforehand, can change over time and can be both human and software agents developed by different parties (Hewitt 1986). Examples of such open systems include computer games, virtual travel agents, E-Commerce solutions etc. The 3D Electronic Institutions provide ways to control the security aspects of those systems and offer mechanisms of their automatic visualization in 3D Virtual Worlds. Figure 1 presents the methodology for achieving this visualization. On the first phase the real world open system is selected to be implemented as an institution. It should clearly exhibit the need for institutional presence: allow interactions with complex protocols and express a need for strong security control. From the above mentioned examples we see virtual travel agents (Bogdanovych, Berger, Simoff & Sierra 2006) and various E-Commerce solutions (Bogdanovych, Berger, Sierra & Simoff 2004) as potential applications. Next, the institution is formalized and mapped onto a 3-dimensional space.

In order to formalize an institution we utilize the Electronic Institutions methodology (Esteva 2003), which helps to structure the interactions between different components (participants) by imposing, and enforcing, well established conventions based on organizational principles. The enforcement of organizational conventions is achieved by separating patterns

Copyright ©2007, Australian Computer Society, Inc. This paper appeared at the Thirtieth Australasian Computer Science Conference (ACSC2007), Ballarat, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 62. Gillian Dobbie, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

¹A property of an object, or a feature of the immediate environment, that indicates how to interface with that object or feature. The empty space within an open doorway, for instance, affords movement across that threshold. A couch affords the possibility of sitting down on it.

²3D character representing a participant in a Virtual World

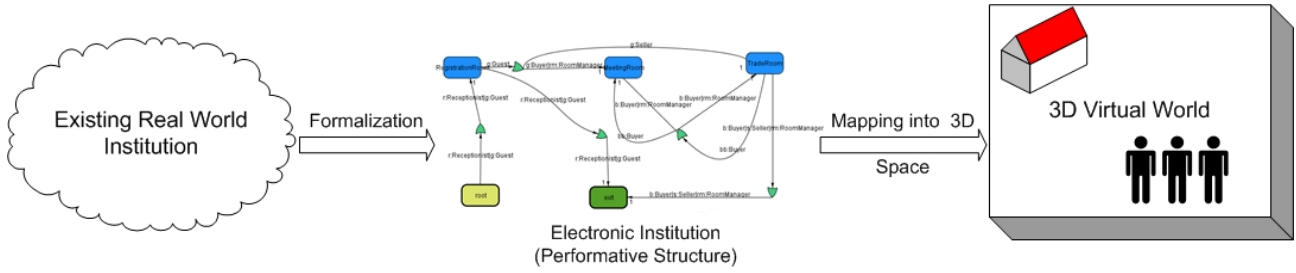


Figure 1: The Methodology for Visualization of 3D Electronic Institutions

of conversational activities into several methodological entities (scenes), assigning different roles to different types of participants, specifying the rules (protocols) for inter-participant interactions and defining the role flow of participants between scenes. The specification of scenes and the role flow are done in a form of a directed planar graph, where nodes represent scenes and arcs and their labels define the role flow. This graph, which is called *Performative Structure* (Esteva 2003), forms a basis for the visualization of the system in 3D Virtual Worlds (as shown on Figure 1). The nodes are visualized as rooms and arcs are transformed into doors connecting the rooms.

To make the visualization of 3D Electronic Institutions more efficient we propose to transform the Performative Structure graph (an important part of the 3D Electronic Institution Specification) into a 3D Virtual World in a fully automatic way. To our knowledge there are no algorithms being proposed for performing such a transformation. However, there is a technology called rectangular dualization (He 1997). It is used for the transformation of a planar graph into a partition of a rectangle R of the plane into subrectangles, where edges of the original graph determine the adjacency of the generated rectangles. Moreover, rectangular dualization offers a space optimal solution for such transformations. However, not all planar graphs admit a rectangular dual, a fact that precludes most applications of the method. In order to overcome this drawback, we developed an algorithm which transforms each planar graph into a new graph by inserting into it a minimum number of new vertices and edges for forcing it to admit a rectangular dual. In the particular case of 3D Electronic Institutions the classical rectangular dualization algorithm is used to create a map of the institution, which is further transformed into a 3D Virtual World. The map is very close to the rectangular dual of the graph representing the adjacencies defined between each pair of rooms belonging to the same floor of the building. In this sense the rectangular dual exploits location awareness defined by the frequency of interactions forecasted for the inhabitants of the same floor and can be used for minimizing the distance between two agents that are supposed to have frequent interactions during their everyday activity. As the Performative Structure graph is not necessarily planar, the original algorithm is changed to firstly remove the intersecting arcs of the source graph forcing it being planar. We also extended the algorithm in such a way that the vertex peers related to removed arcs are stored, and after the generation of the 3D virtual environment the interconnected *teleports* are placed in the corresponding rooms.

The 3D Electronic Institutions, on the one hand, introduce structured interactions into the chaotic nature of Virtual Worlds. On the other hand, the model we propose allows semiautomatic generation of 3D Virtual Worlds from Electronic Institutions specification and runtime maintenance of those worlds. This

will increase the speed of Virtual Worlds development in the future and make online communities more secure. Another technical benefit of this approach is that 3D Virtual Worlds provide general and very intuitive way for human navigation and interaction within. This means that once the 3D Electronic Institution is generated - there is no need for any additional software code to be written to enable humans to start interacting. Furthermore, 3D Electronic Institutions offer a first formal methodology for the design of 3D Virtual Worlds.

The remainder of the paper is structured as follows. Section 2 describes the 3D Electronic Institutions concept. Design considerations for 3D Virtual Worlds are presented in Section 3. The motivation for visualizing 3D Electronic Institutions in an Euclidean way are given in Section 4. In Section 5 the generic description of the rectangular dualization method is given. Section 6 explains how the rectangular dualization can be adapted to the generation of 3D Electronic Institutions. Concluding remarks and details of future work are presented in Section 7.

2 3D Electronic Institutions

3D Electronic Institutions is a concept that appeared from the combination of the Electronic Institutions and 3D Virtual Worlds technologies. This combination resulted in a working methodology, supported by a number of tools, for designing highly secure and reliable immersive 3D solutions. Applying 3D Electronic Institutions methodology requires 4 important steps to be accomplished:

1. Specification of an Electronic Institution using ISLANDER (Esteva, de la Cruz & Sierra 2002).
2. Annotation of the Electronic Institution specification with components of the 3D Virtual World.
3. Automatic generation of the corresponding 3D environment.
4. Integrating the 3D Virtual World into the institutional infrastructure.

On the specification stage the institutional regulations are defined. An institution is seen as an infrastructure for regulating the interactions of autonomous agents, which can be either humans or autonomous software modules (further referred as autonomous agents). The notion of institutions introduces a dramatic difference to the development of multiagent systems compared to the majority of present solutions. Instead of focusing on the implementation details of each particular agent, a system-oriented view is taken. We assume that participating agents may be heterogeneous and self-interested, and we cannot rely on their correct behavior. Therefore, the institution is designed as a set of limitations which every participant have to comply with. This assumption permits

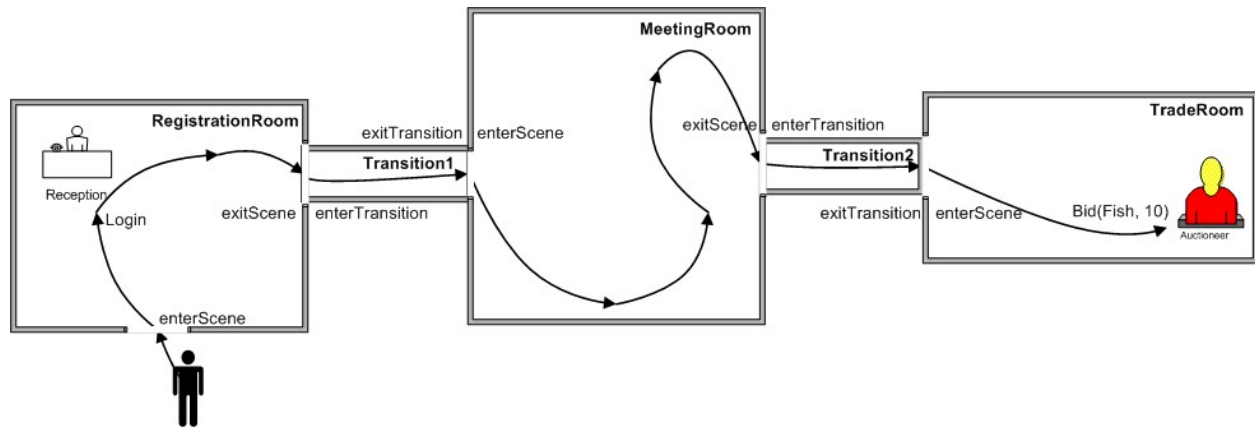


Figure 2: An example of a simple institution.

that agents behave autonomously and make their decisions freely up to the limits imposed by the institution. The rules about user behavior in an institution are introduced on the specification phase determined by three types of conventions:

- Conventions on language, the *Dialogical Framework*. Determines what language ontology and illocutionary particles agents should use. It also fixes the organizational structure of the society of agents, that is, which roles agents can play, and what the incompatibilities and relationships among the roles are.
- Conventions on activities, the *Performative Structure*. This dimension determines in which types of dialogues agents can engage. Each different activity an agent may perform is associated to a dialogue among the group of agents involved in that activity. These (structured) dialogues are called *scenes*. The Performative Structure fixes which protocol (possible dialogues) can be enacted in each scene, which sub-language of the overall institutional language can be used in each scene, and which conventions regulate the in and out flux of agents in scenes. Finally, the minimum and maximum number of participants is limited by the specification of scenes. Scenes are interconnected to form a network in order to represent sequence of activities, concurrency of activities or dependencies among them. Agents leave scenes where they have been playing a given role and enter other scenes to play the same or a different role. This transit of agents is regulated by special (simple) scenes called *transitions*. Transitions re-route agents and are where synchronization with other agents (if needed) takes place. Sometimes new scenes can only be enacted by a group of agents, or agent can only join scenes as members of a group. Transitions are the places where agents synchronize before moving on.
- Conventions on behavior, the *Norms*. Institutions impose restrictions on the agents actions within scenes. These actions are basically restricted to: illocutions and scene movements. Norms determine the commitments that agents acquire while talking within an institution. These commitments restrict future activities of the agent. They may limit the possible scenes to which agents can go, and the illocutions that can henceforth be uttered.

While the specification strictly defines the limitations, it also helps to understand what participants

need in order to operate in the institution. Some elements of the specification have conceptual similarities with building blocks in 3D Virtual Worlds, which makes it possible to automatically generate a 3D representation of the specification. The scenes and transitions, for example, are transformed into 3D rooms, connections correspond to doors, and the number of participants allowed in a scene determines the size of a room (see (Bogdanovych, Berger, Sierra & Simoff 2005) for details). To make the resulting 3D Virtual World more appealing, the specification is annotated with additional 3D related components (objects, textures, animations etc). After accomplishing this step the generated 3D Virtual World is ready to be visualized and the 3D Electronic Institution infrastructure will be executed to take care of the validity of interactions between participants, verify the permissions of participants to access different scenes and will make sure that all the institutional norms and obligations are imposed.

3D Electronic Institutions have 2 different levels of execution: institutional level and social level. The institutional level makes sure that the institutional rules are not violated. On this level a participant is restricted to sending a text message to the institution for verification, requesting to perform an action. If under the current circumstances the action is allowed to be performed without violating the rules, the agent receives back a response and the action is visualized. The way the actions are visualized can not be changed, as well as their execution can not be terminated. For human participants sending text messages is transparent as it happens as a result of their actions in the Virtual World.

Actions that are not controlled by the institution are performed at the social level. These actions are executed directly by the participant without prior verification by the institution. For example, there is no need for the institution to specify how the participants should walk from one room to another. Figure 2 presents an example of a simple institution and shows the actions performed on both social and institutional level. The institutional level actions include: (enterScene, exitScene, enterTransition, exitTransition and login). On the social level these are: moving, clicking, colliding, rotating etc. The black arrows on the picture show the trajectory of the participant's movement through registrationRoom, meetingRoom to TradeRoom. The black figure represents the participant, other figures correspond to internal agents (employees of the institution) Receptionist and Auctioneer. The Receptionist welcomes the participant in the RegistrationRoom, verifies the login and password and unlocks the doors to other scenes if the

identity of the participant is proven. The Auctioneer sales different goods in the TradeRoom. It announces the current product to be auctioned, waits for incoming bids and sells it to the winner of the auction. The MeetingRoom is used for social interaction between buyers.

3 Design Considerations for 3D Virtual Worlds

Human beings live in a well structured space following different metaphors. Popular everyday metaphors, such as *buildings*, *streets*, *landscapes* etc. are widely used in Virtual Worlds (Russo Dos Santos, Gros, Abel, Loisel, Trichaud & Paris 2000). We adopt this class of metaphors for the visualization of 3D Electronic Institutions. A visual representation of 3D Electronic Institutions is mapped onto the metaphor of a small town. Each building constitutes a separate institution, public transport might be used to access different institutions, rooms relate to different activities that can be performed in the institution.

Virtual Worlds are spaces where people “meet”. Social interaction is a key feature and Virtual Worlds have to provide support for communication and collaboration of their participants (Smith, Maher & Gero 2003). Furthermore, not all the inhabitants of Virtual Worlds are under human control. Some of the participants are autonomous agents that have an embodied representation. Thus, a Virtual World, which is populated by automated embodied agents and agents driven by human beings, has to take care of their different *abilities*.

During the construction of a 3D representation of a virtual environment it is important to keep the benefits of traditional 2D interface design in mind (Bowman, Kruijff, LaViola & Poupyrev 2001). Participating in a 3D environment in which users can manipulate 3D objects, doesn't necessarily mean exclusion of 2D user interface elements. In fact, the interaction with 2D interface elements offers a number of advantages over a 3D representation for some tasks. Most efficient selection techniques, for instance, are widely realized in 2D, whereas, the selection process in a 3D user interface must consider the *user's viewpoint and distance* to the object. Combining the advantages of 2D and 3D design is a very powerful and intuitive approach for the construction of Virtual Worlds.

Besides the benefits obtained by adding an additional dimension for visualization purposes, this new degree of freedom introduces new difficulties (Nielsen 1998). More precisely, not every application domain has a suitable and usable representation in a three-dimensional way. The more abstract (the more non-physical) an application domain becomes, the harder it gets to visualize it in 3D. Consider for example, a 3D representation of a hyperspace like the World Wide Web. Navigating through a three-dimensional representation of web sites will end up in a rather confusing task for most users. Thus, it is of great importance to consider carefully whether to use the “third dimension” for the particular application.

In our approach we take advantages from both ways of representation. The visualization of 3D Electronic Institutions contains 3D elements (3D Virtual World itself) and 2D Elements (map of the institution and the backpack which helps to remind user's obligations towards the institution).

4 Euclidian Representation

Navigation is an important issue in the design of 3D virtual environments. Navigational problems may

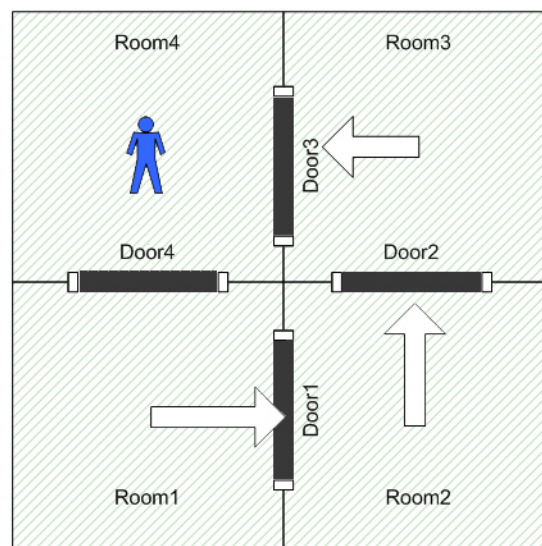


Figure 3: Euclidian representation. Human knows that Room 1 must be behind Door 4.

break the immersive experience and lead to the rejection of the system by end users. Humans live in an Euclidian world: distances and angles help humans to navigate it. In our opinion, the better our system imitates real world, the more it supports social factors like communication and collaboration.

3D Virtual Worlds could certainly be programmed without an Euclidian model in mind. For instance, we could create a series of rooms interconnected by teleportation³. In this case the distance between each two connected rooms will be equal to zero. Technically, such a solution poses much less problems for development, as there is no need to control the positioning of every room. The simplest implementation solution here would be to have each room stored as a separate file, which is loaded on demand when user wants to teleport into the corresponding room.

Despite the technological simplicity of the non-Euclidean way of representing an institution as a Virtual World, such an approach may cause navigational problems for the participants inside. Moreover, we believe that having an Euclidian representation of the Electronic Institution helps in learning of the institutional structure.

To demonstrate the confusion a participant may experience navigating a non-Euclidean Virtual World consider the situation (Figure 3) where the user walks from Room 1 to Room 4 following the arrows. In Room 4 the human should correctly expect that Room 1 is behind Door 4, otherwise the believability of the interface will be lost. This expectation of Room 1 is based on both the consistency of the navigational layout and intuitive feeling about the size of visited rooms.

Unfortunately, the problems of Euclidean and non-Euclidean visualizations of Virtual Worlds are understudied. Despite this fact, there is some research evidence in favor of our initial hypothesis that motion techniques which instantly teleport users to new locations are correlated with increased user disorientation. In (Bowman, Koller & Hodges 1997) the authors present the results of their user study, where one of the questions was whether teleportation can cause navigational problems. The study clearly shows that teleportation (or jumping technics, as it is called in

³the process of moving users from one place to another more or less instantaneously, without passing through the intervening space

the paper) can reduce the user's spatial awareness. With teleportation there is no sensation of motion, only that the world has somehow changed around the user. It is a technique whose motion has no analog in the physical world. Moreover, authors came to a conclusion that frequent teleportation may even reduce the sense of presence in a Virtual World, which would eliminate one of the most important benefits of the 3D technology.

Another research (Ruddle 2000) provides some support in favor of the assumption behind the example presented on Figure 3. This user study analyzed the navigational efficiency of participants in the overlapping Virtual Worlds. The results suggest that some users had great difficulties navigating them. Even the experimenter, who informally observed participants as they travelled through the Virtual Worlds, was often unsure of which door to go through to enter a particular room.

In order to avoid the difficulties with navigation and spacial awareness of participants we propose an automatic technique for generation of an Euclidean representation of a 3D Electronic Institution. In our case this Euclidian representation is generated from the graph representing the performative structure. To achieve this, we propose to adapt *rectangular dualization* technology. The algorithm presented in Section 5 removes the intersecting arcs of the source graph to force it being planar. We extend the algorithm in such a way that the vertex peers related to removed arcs are stored, and after the generation of the 3D virtual environment the interconnected teleports are placed in the rooms represented by those vertices. Although the teleports will still be present, the number of them is supposed to be insignificant to pose any navigational problems.

5 Rectangular Dualization

Rectangular dualization was originally used to generate rectangular topologies for floor planning of integrated circuits: by a floor plan, we partition a rectangular chip area into rectilinear polygons corresponding to the relative location of functional entities of the circuit (He 1997). In spite of the specialized problems that motivated its origin, rectangular dualization contributes to the resolution of many other visualization problems having in common with circuits the condition that objects and their interoccurring relations are represented by means of a planar graph. An example is given by network configuration issues, when human interventions of design or topology adjustment are needed and a physical or logical layout representation becomes essential for the human operator.

In fact, a very serious problem to cope with in graph drawing is how to represent edges in such a way that they do not appear too close together. The aim is to enhance the readability of the drawing, making easier to find out which nodes are connected by an edge. The very first solution to this problem is to avoid edge crossings, and this motivates the interest for *planar graphs*, that are precisely those graphs that can be drawn in the plane with no edge crossings. The choice for planar graphs is not only a representation facility but is primarily validated by real-world examples where the presence of crossing links may produce technical drawbacks.

Further on, since a major optical effort is encountered in the proximity of vertices, where adjacent edges need to meet in a point, several studies have been spent in devising drawing algorithms capable of maximizing *angular resolution*, i.e. the smallest angle between adjacent edges, in such a way that lines representing connections are kept as separate as possible;

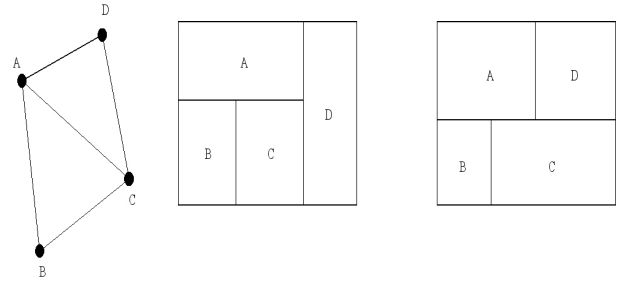


Figure 4: A planar triangulated graph and two possible rectangular layouts.

rectangular dualization results to be an effective visualization method since only orthogonal lines occur.

Thomassen proved that every planar graph is the intersection graph of a collection of three-dimensional boxes, with intersections occurring only in the boundaries of the boxes. Furthermore, he characterized the graphs that have such representations (called *strict representations*) in the plane. These are precisely the proper subgraphs of 4-connected planar triangulations. Together with earlier work (Thomassen 1984), his work yields an algorithm for testing a graph G to see if it admits a rectangular dual and, if so, constructing such a representation. His proof does not look to lead to an efficient algorithm, however: for a graph having n nodes, a straightforward implementation of his method requires at least $O(n^3)$ time. Bhasker and Sahni (Bhasker & Sahni 1988) developed linear-time algorithms to find a rectangular dual for graphs satisfying Kozminski and Kinnen criterion. In (Kant & He 1993) Kant and He explain how to construct a rectangular dual from a *regular edge labeling* (REL, for short) and present two algorithms to compute such labeling, one based on an edge contraction technique and the other on a *canonical ordering*. A later work of Saidur presents a linear time algorithm which finds a rectangular grid drawing using a depth first search (Md. Saidur 1999). In (Buchsbaum, Gansner, Procopiuc & Venkatasubramanian n.d.) is shown how the works above may be combined to produce an efficient algorithm for constructing rectangular duals with asymptotically bounded area. Lai and Leinwand (Lai & Leinwand 1988) first presented the idea of forcing rectangular dual admissibility by introducing crossover vertices breaking all separating triangles. They conjectured that finding a minimal set of crossover vertices was a NP -complete problem and performed non optimal introduction of crossover vertices in linear time. Ancona et al. (Accornero, Ancona & Varini 2000) showed that all separating triangles can be optimally broken in polynomial time and presented an asymptotical bound of $O(n^3)$, which some discussion can refine up to $O(mn \lg n)$, being m the number of edges and n the number of nodes of the input graph. In the following section we describe an implementation that transforms graphs that do not admit rectangular duals into graphs admitting one by adding the minimum number of new vertices.

5.1 Definitions

A *rectangular dual* of a planar graph $G = (V, E)$ is a rectangle R with a partition of R into a set $\Gamma = \{R_1, \dots, R_n\}$ of non overlapping rectangles such that:

- no four rectangles meet at the same point;
- there is a one-to-one correspondence $f : V \rightarrow \Gamma$ such that two vertices u and v are adjacent in G

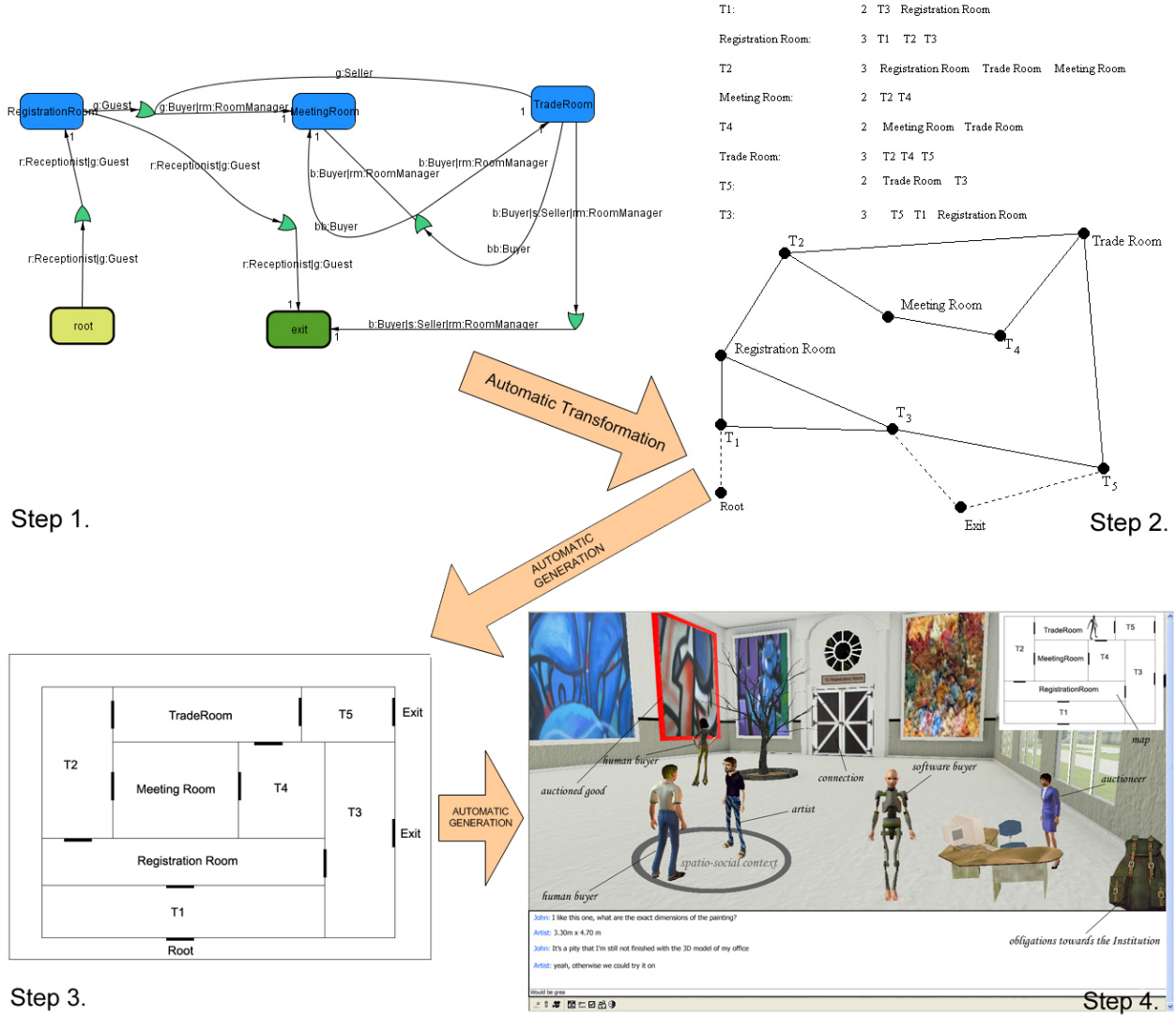


Figure 5: Generating the 3D representation of a Performative Structure graph.

if and only if their corresponding rectangles $f(u)$ and $f(v)$ share a common boundary.

It is easy to see that if a graph admits a rectangular dual, it may not be unique (see Figure 4).

On the other hand, some graphs do not admit rectangular dual. Kozminski and Kinnen present necessary and sufficient conditions under which a plane graph G has a rectangular dual (Kozminski & Kinnen 1984): the most important point is the absence of *separating triangles* (i.e. 3-vertex cycles with at least one vertex in their interior), a condition that in planar triangulations is equivalent to *4-connectivity* whose meaning is that the removal of any set of 3 vertices leaves the remainder of G connected. A *matching* in G is a subset M of edges such that for every vertex v , at most one edge e covers v , that is v is an endvertex of e . A graph is *k-regular* if every vertex has degree k , that is k incident edges. A *maximum matching* is a matching with largest possible cardinality. If the graph is weighted, we may even consider a *maximum weight matching*. A *bridge* is an edge whose removal disconnects G . Whenever we speak of a planar graph, we assume that some planar embedding has been fixed, which corresponds to the idea of depicting an existent physical connection among real objects (in this perspective, it would be more accurate to speak of *plane graphs*, i.e. planar graphs with a fixed embedding in the plane). A *structured*

graph is a form of abstraction applied to a large graph in order to make it modular and more manageable. The abstraction consists in collapsing a subgraph to a single vertex (called a *macrovertex*), or to a single link (called a *macrolink*) thus obtaining a simpler and hierarchically described graph. The structuring operation is usually iterated recursively until a large graph is decomposed into relatively small and manageable components and sub-components defined at several levels of nesting, adopting a methodology that is usually applied to every large project (software and hardware design) involving hundreds or thousands of components.

6 Implementation Details

To visualize the 3D Electronic Institutions we use the metaphor of architecture. Each institution is visualized as a building, and each building consists of different rooms. This approach helps to provide users with the interface close to the users' everyday life surroundings.

From the Performative Structure graph, a highly important part of the specification, both the 3D representation of the institution and the map of the institution are created. The process of automatic generation of a 3D Virtual World and a map from this graph is depicted in Figure 5.

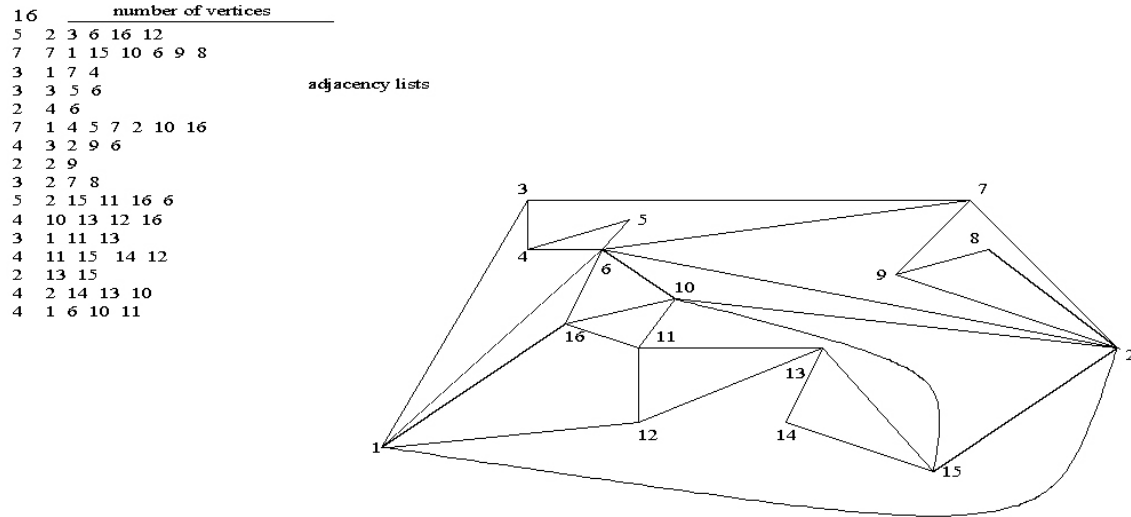


Figure 6: A planar embedding of a graph with its input file.

In the upper left corner in Figure 5 the source Performative Structure graph is presented. This graph corresponds to the simple institution from Figure 2. Rectangular shapes represent scenes, triangular shapes are transitions, arcs connecting the nodes are connections. As it was already mentioned earlier, the scenes and transitions are transformed into rooms, and the connections are visualized as doors. In this example the 3D Electronic Institution consists of 3 functional rooms: registration room, meeting room and trading room. These 3 rooms are contained inside a building and the building itself is placed into a garden. Root and Exit scenes are not functional, they only determine entrance and exit points of the institution, so from the 3D Virtual Worlds prospective entering these rooms will force exiting the institution building and moving into the garden.

The automatic generation is done in 4 steps.

On *Step 1* the redundant information contained in the Performative Structure is filtered out. If some nodes of the graph, for example, are connected with more than one arc only one randomly selected arc is left and all the others are deleted. Next, the performative structure graph is transformed into a form accessible by the OCoRD software. The OCoRD (Optimal Constructor of a Rectangular Dual) software is a tool aiming at the solution of floorplanning problems and at the orthogonal drawing of planar networks. Given a planar embedding of the graph, it accepts a numeric adjacency lists and, if the graph admits a rectangular dual, it returns its coordinates in the plane and a drawing in fig format.

On *Step 2* OCoRD transforms a graph not admitting a rectangular dual into a 4-connected supergraph satisfying Kozminski and Kinnen criterion and creates its rectangular dual.

In order to force an input graph to satisfy the Kozminski and Kinnen criterion it is necessary to eliminate separating triangles by adding crossover vertices on an edge of each separating triangle (Lai & Leinwand 1988). The breaking method implemented in OCoRD is optimal, in other words it only adds a minimum number of such crossover vertices and inserts them in strategical positions, such that a single vertex may break two triangles or more (Accornero et al. 2000). OCoRD performs this transformation in the following steps:

1. four external vertices are added according to the construction presented in (Kant & He 1993);

2. the geometrical dual of the graph is computed and faces belonging to a separating triangle are detected and clustered together in a single macrovertex;
3. a covering affecting macrovertices is computed; the effect is that all separating triangles are optimally broken by inserting new vertices in some strategical places along some of their constituting edges;
4. the resulting graph is triangulated with the algorithm described in (Biedl, Kant & Kaufmann 1997).

The result of this transformation is a graph satisfying triangularity and four-connectivity. Faces belonging to a separating triangle are detected in linear time (Chiba & Nishizeki 1985). Separating triangles are broken by solving a minimum unweighted macro-covering problem on the geometrical dual of input graph. The macro-covering can be computed by solving a sequence of minimum *weighted* edge covering problems on each simple graph of the structured dual. In (Parekh 2002), Parekh showed how to reduce a minimum weighted edge cover of a specified subset of the vertices of G to a maximum weighted b -matching, a well solved problem (Edmonds & Johnson 2003) that is worked out by implementing the $O(mn \lg n)$ algorithm presented in (Galil 1986). The resulting graph (Figure 6) is biconnected and it is described through adjacency lists satisfying the following properties:

- vertices are indexed by non negative integers $(1, \dots, n)$;
- for each internal vertex, its adjacencies are clockwise listed, according to a fixed planar embedding of the graph, starting from the vertex having the lowest index;
- for each vertex belonging to the graph contour, its adjacencies are clockwise listed, starting from the vertex on the contour which precedes it in a counterclockwise direction with respect to the contour.

In the modification perspective, also input graphs that are not under the biconnection constraint may be processed: a preliminary step can be implemented to provide this degree of connectivity (Read 1987).

Algorithm: Rectangular Dual Construction

Input Planar biconnected graph G
Output A rectangular dual of G

Add four external vertices
 Compute the geometrical dual graph G^*
 Detect faces belonging to a separating triangle, for any
 Collapse them into a macrovertex in G^*
 Solve the macro-covering problem in G^*
 Add new vertices in G
 Triangulate G (Biedl et al. 1997)
 Compute a REL
 Compute the rectangular dual coordinates
 Delete external vertices and draw

Figure 7: Algorithm: Rectangular Dual Construction

The rectangular dual of the transformed graph is produced next. Figure 7 gives an overview of the implemented rectangular construction method.

The result of this algorithm needs to be postprocessed to remove the rectangles, which were introduced because of the breaking points and to reshape the adjacent rooms to the size of removed rectangles. But not all those rectangles are removed. Note, that the Root and Exit scenes are always present in the performative structure graph. Moreover the root scene is not permitted to have incoming arcs and the exit scene doesn't have any outgoing arcs. As those scenes are not visualized anyway, the corresponding graph nodes were assigned as two of the four external vertexes. In this way the garden is automatically created as the rectangle surrounding the graphs rectangular dual.

Now the only thing that is left - is placing the doors between connected rooms. The outcome of this step is the map of the institution, which is presented in the lower left corner of Figure 5.

As a refinement of the above rectangular dual construction method, we may say that its logarithmic cost is due to the fact that the aforementioned matching algorithm holds for general graphs, a much wider class of graphs than the one we deal with in our planarity assumption. Instead, a matching in a 3-regular bridgeless graph can be found in linear time (Biedl, Bose, Erik D. Demaine & Lubiw 1999). Since the collection of all planar 3-regular bridgeless graphs is exactly the collection of duals of planar triangulations where the outside face is a triangle, we may tighten the bound by solving the matching problem on the dual of a planar triangulation and this can be obtained by producing a triangular outer boundary and by running the algorithm (Biedl et al. 1997) (which triangulates without adding new separating triangles) before the computation of the geometrical dual.

Step 3, transforming a 2D map of the institution into a 3D Virtual World, is pretty much trivial. The coordinates of the map are first transformed into the coordinates of the 3D Virtual World. Then every room is scaled so that it can physically contain the maximum number of participants that is defined for it. Later on the corresponding 3D objects are reshaped and put into the 3D Virtual World. The result will look similar to the bottom right part of the Figure 5.

On *Step 4* the resulting 3D Virtual World is annotated then visualized.

7 Conclusions and Future Work

We presented an algorithm for the automatic transformation of a graph into a 3D Virtual World and a

particular application of the algorithm for the automatic generation of the Euclidean representation of 3D Electronic Institutions. The rectangular dualization proved to be a reasonable technique for performing this task, as it provides the needed transformation of the content of a discrete diagram (the Performative Structure graph) into a continuous spatial representation (the 2D map of the Institution). The adopted approach results efficient because in most cases of practical interest its complexity is linear $O(n)$ time, where n is the number of nodes in the input graph.

A new implementation of the Separating Triangles Elimination algorithm is under development. This algorithm reduces the problem to a minimum cost perfect matching problem i.e., a perfect matching of smallest possible cost in the geometrical dual graph. The matching is almost as fast as the best known matching algorithm for the problem without costs, that is maximum cardinality matching. In fact, the geometrical dual graph of a triangulated graph is a 2-connected cubic graph which satisfies the conditions of Peterson's theorem (Biedl et al. 1999), granting that such a graph always admits a perfect matching that can be computed in $O(n)$ time for planar graphs. The new method operates on the original graph (and on its geometrical dual) without structuring it into a hierarchy of nested graphs, thus requiring simpler data structures.

Furthermore, the presented algorithm widens the class of tractable graphs without penalizing the running time and the layout area of the best cases; for the worst ones (general non 4-connected graphs) the running time is at most logarithmic, a computational cost compensated by a rectangular dualization solution that otherwise would be unavailable. Such a solution is obtained by adding the minimum number of vertices in order to complete the graph up to 4-connectivity, an approach which is respectful of the area minimization drawing criterion.

Another advantage is that rectangular dualization, with its multi-scale capability of containing nested rectangles, has a predisposition for the dynamic drawing of hierarchically organized 3D Electronic Institutions or the possibility of incrementing the layout visualization at subsequent steps of the navigation. In fact, hierarchical generation and visualization of a 3D Electronic Institution is naturally embeddable into a hierarchy of rectangular dual graphs. Then, the construction of the dual of a hierarchically organized Performative Structure graph can be performed in two ways: by recursively applying the construction to each graph of the hierarchy and by forcing the rectangles representing a cluster in the plain graph (they must be adjacent) to form a single rectangle.

References

- Accornero, A., Ancona, M. & Varini, S. (2000), 'All Separating Triangles in a Plane Graph Can Be Optimally "Broken" in Polynomial Time', *International Journal of Foundations of Computer Science* **11**(3), 405–421.
- Bhasker, J. & Sahni, S. (1988), 'A Linear Algorithm to Find a Rectangular Dual of a Planar Triangulated Graph', *Algorithmica* **3**, 247–278.
- Biedl, T. C., Bose, P., Erik D. Demaine, E. D. & Lubiw, A. (1999), Efficient Algorithms for Petersen's Matching Theorem, in 'Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms', N.Y., pp. 130–139.
- Biedl, T. C., Kant, G. & Kaufmann, M. (1997), 'On Triangulating Planar Graphs Under the Four-Connectivity Constraint', *Algorithmica* **19**(4), 427–446.
- Bogdanovych, A., Berger, H., Sierra, C. & Simoff, S. (2004), 3d electronic institutions: Social interfaces for e-commerce, in 'Proceedings of the 2nd European Workshop on Multi-Agent Systems', Barcelona, Spain.
- Bogdanovych, A., Berger, H., Sierra, C. & Simoff, S. (2005), Narrowing the Gap between Humans and Agents in E-commerce: 3D Electronic Institutions, in 'Proceedings of the 6th International Conference on Electronic Commerce and Web Technologies (EC-Web'05)', LNCS 3590, Springer-Verlag, pp. 128–137.
- Bogdanovych, A., Berger, H., Simoff, S. & Sierra, C. (2006), Travel agents vs. online booking: Tackling the shortcomings of nowadays online tourism portals, in 'Proceedings of the 13th International Conference on Information Technologies in Tourism (ENTER'06)', Springer, Lausanne, Switzerland, pp. 418–428.
- Bowman, D. A., Koller, D. & Hodges, L. F. (1997), 'Travel in immersive virtual environments: An evaluation of viewpoint motion control techniques', *vrais* **00**, 45.
- Bowman, D., Kruijff, E., LaViola, J. & Poupyrev, I. (2001), 'An introduction to 3-d user interface design', *Presence: Teleoperators and Virtual Environments* **10**(1), 75–95.
- Buchsbaum, A. L., Gansner, E. R., Procopiuc, C. M. & Venkatasubramanian, S. (n.d.), 'Rectangular layouts and Contact Graphs', pp. 1230–1234. submitted paper.
- Chiba, N. & Nishizeki, T. (1985), 'Arboricity and Subgraph Listing Algorithms', *SIAM Journal on Computing* **14**(1), 210–223.
- Edmonds, J. & Johnson, E. L. (2003), Matching: A Well-Solved Class of Integer Linear Programs, in M. Jünger, G. Reinelt & G. Rinaldi, eds, 'Combinatorial Optimization: Eureka, You Shrink!', Papers Dedicated to Jack Edmonds', LNCS 2570, Springer, pp. 27–30.
- Esteva, M. (2003), Electronic Institutions: From Specification to Development, PhD thesis, Institut d'Investigació en Intel·ligència Artificial (IIIA), Spain.
- Esteva, M., de la Cruz, D. & Sierra, C. (2002), Islander: an electronic institutions editor, in 'First International Conference on Autonomous Agents and Multiagent systems', ACM Press, Bologna, pp. 1045–1052.
- Galil, Z. (1986), 'Efficient Algorithms for Finding Maximum Matching in Graphs', *ACM Computing Surveys* **18**(1), 23–38.
- He, X. (1997), On Floorplans of Planar Graphs, in 'Proceedings of the 29th Annual ACM Symposium on the Theory of Computing (STOC '97)', ACM Press, pp. 426–435.
- Hewitt, C. (1986), 'Offices are open systems', *ACM Transactions on Office Information Systems* **4**(3), 271–287.
- Kant, G. & He, X. (1993), Two algorithms for finding rectangular duals of planar graphs, in J. van Leeuwen, ed., 'Computer Science (WG'93)', LNCS 790, Springer, Utrecht, The Netherlands, pp. 396–410.
- Kozminski, K. & Kinnen, E. (1984), An Algorithm for Finding a Rectangular Dual of a Planar Graph for Use in Area Planning for VLSI Integrated Circuits, in 'ACM IEEE 21st Design Automation Conference (DAC '84)', LNCS 3590, IEEE Computer Society Press, Los Angeles, Ca., USA, pp. 655–656.
- Lai, Y.-T. & Leinwand, S. M. (1988), 'Algorithms for Floorplan Design Via Rectangular Dualization', *IEEE Transaction on Computer-Aided Design* **7**, 1278–1289.
- Md. Saidur, R. (1999), Efficient Algorithms for Drawing Planar Graphs, PhD thesis, Department of System Information Science, Graduate School of Information Sciences, Tohoku University, Japan.
- Nielsen, J. (1998), '2d is better than 3d', <http://www.useit.com/alertbox/981115.html>. visited 22 June 2004.
- Parekh, O. (2002), Edge Dominating and Hypomatchable Sets, in 'Proceedings of 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'02)', ACM/SIAM, San Francisco, CA, USA, pp. 287–291.
- Pine, B. I. & Gilmore, J. (1998), 'Welcome to the experience economy', *Harvard Business Review* pp. 97–105.
- Read, R. C. (1987), 'A New Method for Drawing a Graph given the Cyclic Order of the Edges at Each Vertex', *Congr. Numer.* **56**, 31–44.
- Ruddle, R. A. (2000), Navigating overlapping virtual worlds: Arriving in one place and finding that you're somewhere else, in 'Spatial Cognition II, Integrating Abstract Theories, Empirical Studies, Formal Methods, and Practical Applications', Springer-Verlag, London, UK, pp. 333–347.
- Russo Dos Santos, C., Gros, P., Abel, P., Loisel, D., Trichaud, N. & Paris, J. P. (2000), Mapping Information onto 3D Virtual Worlds, in 'Proceedings of the International Conference on Information Visualization', pp. 379–.
- Smith, G. J., Maher, M. L. & Gero, J. S. (2003), Designing 3D Virtual Worlds as a Society of Agents, in 'Proceedings of CAADFutures'.
- Thomassen, C. (1984), 'Plane Representations of Graphs', *Progress in Graph Theory* pp. 43–69.

QUIP: A protocol for securing content in peer-to-peer publish/subscribe overlay networks

Amy Beth Corman*

Peter Schachte*

Vanessa Teague

*National ICT Australia, Victoria Lab
 Department of Computer Science & Software Engineering
 The University of Melbourne
 Email: {amy,schachte,vteague}@csse.unimelb.edu.au

Abstract

Publish/subscribe networks provide an interface for publishers to perform many-to-many communication to subscribers without the inefficiencies of broadcasting. Each subscriber submits a description of the sort of content they are interested in, then the publish/subscribe system delivers any appropriate messages as they are published. Although publish/subscribe networks offer advantages over traditional web-based content delivery, they also introduce security issues. The two security problems that we solve are: ensuring that subscribers can authenticate the messages they receive from publishers, and ensuring that publishers can control who receives their content. We propose QUIP, a protocol which adds efficient authentication and encryption mechanisms to existing publish/subscribe overlay networks. The idea is to combine an efficient traitor-tracing scheme (by Tzeng and Tzeng (2001)) with a secure key management protocol. This allows publishers to restrict their messages to authorised subscribers and to add and remove subscribers without affecting the keys held by the other subscribers.

Keywords: Peer-to-peer, publish/subscribe, security, network protocol

1 Introduction

There are many variations on content delivery systems used by the Internet today, each with its own advantages and disadvantages. We propose using a peer-to-peer publish/subscribe model to efficiently manage and securely deliver intermittent content. We will use web comics, the dissemination of electronic comic strips, as an illustrative example throughout this paper.

There are many advantages to be gained from using a publish/subscribe system as opposed to the current model of delivering web comics via web pages. One advantage to subscribers is that a push model is more convenient than a pull model because it is not necessary for subscribers to take action in order to have their content delivered. A push model is especially beneficial if publications are made at unpredictable intervals since the subscriber will never waste time looking for something that is not there. A subscriber may use the same interface to access multiple web comics from different publishers. A publish/subscribe system enables the possibility of a single billing interface for these to the subscriber. This

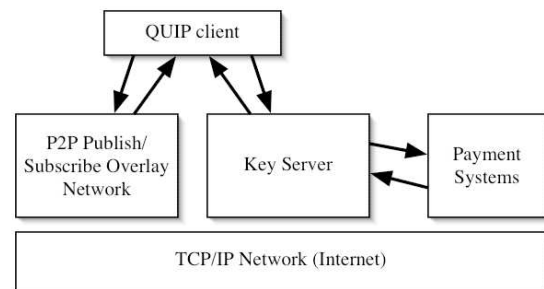


Figure 1: An illustration of the relationship of the network layers.

has multiple benefits. It is simpler for the subscriber to only enter payment information in one place, and only requires the subscriber to trust the payment authority and not each individual publisher. This allows the subscriber to enjoy content from any publisher without needing to worry about the safety of their payment details. Centralised payment improves efficiency since only one organisation handles payment leaving publishers free to focus on creating new content. This will enable new publishers to gain subscribers without needing a reputation for trustworthiness. Publish/subscribe systems also provide advantages in matching publishers to interested subscribers.

A publish/subscribe system provides further advantages for publishers. In the current web delivery method, publishers depend on advertisers for income, which means that the subscribers must view ads on the same page as the comic. This system allows the publishers to charge the subscriber directly. The publishers do not need to maintain a constant presence on the Internet and need only connect long enough to transmit the publication into the publish/subscribe network.

We introduce QUIP, a protocol for secure content distribution in peer-to-peer publish/subscribe overlay networks. The goal of QUIP is to securely provide all the advantages of a publish/subscribe distribution system. Our approach is to add flexible and efficient encryption and authentication mechanisms to existing publish/subscribe systems.

A content-based publish/subscribe system works as follows. A publisher must advertise the details of their expected publications. In the context of web comics, this could include the name of the comic, a category of the type of comic and any other descriptive data the publisher would like to provide. A subscriber then decides what comics they are interested in and sets up a filter that will match events as described by the publisher (in the simplest case, this would be the title of the comic). The publisher then publishes events (such as a single comic) which are routed through the publish/subscribe infrastructure

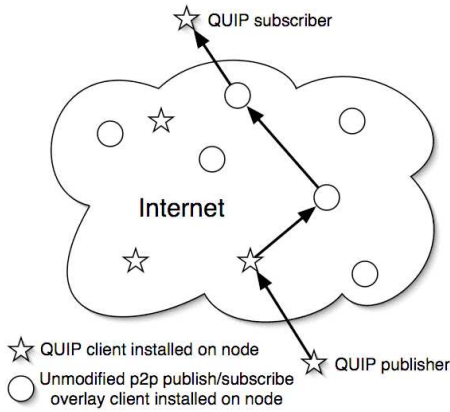


Figure 2: An illustration of p2p publish/subscribe overlay layer of the network.

to the interested subscribers.

The relationship between our protocol and the existing network systems is shown in Figure 1. We create a *key server* which is separate from the publish/subscribe overlay network. We also add a *QUIP client* which interacts with both the publish/subscribe network and the key server. The key server does not interact with the publish/subscribe overlay directly. The QUIP key server will also interface with existing *payment systems*. It is not necessary for all nodes in the publish/subscribe network to run our client, as illustrated in Figure 2.

QUIP is designed for applications with a large number of subscribers and a smaller number of publishers. A single entity may be both a subscriber and publisher. A subscriber may subscribe to multiple publications and a publisher may publish multiple publications.

QUIP uses a public key traitor tracing scheme proposed by Tzeng and Tzeng (2001) (described in Section 4.6) which provides the ability to add and remove subscribers by updating only the publication key. Each subscriber is given a unique key in this scheme which makes it possible to determine which user has leaked their key if a key is posted publicly.

The rest of this paper is organised as follows: Section 2 discusses related work, Section 3 discusses our threat model and requirements, Section 4 describes the QUIP protocol in detail, Section 5 comments on performance and Section 6 concludes and suggests some further work.

2 Related Work

We have similar but not identical goals to Eventguard (Srivatsa & Liu 2005). Both Eventguard and QUIP are systems which sit on top of existing publish/subscribe overlay networks. Eventguard has six main components which are designed to protect each of the five major publish/subscribe actions (subscribe, unsubscribe, publish, advertise, unadvertise) plus routing. Eventguard uses ElGamal for encryption, signatures and the creation of tokens. The major differences between our two solutions are that we do not assume private channels and we do not put so much emphasis on the privacy of subscriptions. Eventguard needs to update subscriber topic keys whenever one subscriber leaves, while this is not necessary with QUIP.

Although Srivatsa and Liu claim Eventguard does not depend on the underlying IP network to provide confidentiality, integrity or authenticity, it is hard to see how these properties are provided based on the

examples given. If we look at the subscribe guard as an example (shown in Equation 1) we can see that the key $K(w)$ is being transmitted to the subscriber from the *Replicated Trusted Meta-service* (referred to as the *MS*) and there is no provision for encrypting it or protecting it from eavesdropping.

$$sb(w) = \langle K(w), T(w), sig_{MS}^S(T(w)), UST^S(w) \rangle \quad (1)$$

It is clear that a secure channel is needed between the subscriber and the *MS* to maintain the secrecy of the key. We assume that Srivatsa and Liu intended to use SSL to secure communication between the *MS* and the clients, which would fix the problem of eavesdropping. It is still not clear, however, how re-keying is to be handled securely without client certificates. In Eventguard, each subscriber and topic pair has a unique identifier, but it is not stated how this would be securely mapped to the correct user. Whenever a subscriber unsubscribes, a new topic key must be generated and communicated securely to the remaining subscribers and the publisher. It is also important to note that the traffic generated from re-keying grows quadratically with the group size.

In QUIP, we have chosen to relax the requirement for privacy of subscriptions because we feel that for many applications including web comics only a moderate amount of privacy is necessary. We feel that the mechanisms provided in Eventguard are unnecessary in our context, and are insufficient where privacy is essential. Specifically, with Eventguard the name of topics is obscured but only from random observers, since every subscriber to a particular topic knows the name of that topic, they can identify the name of that topic in any traffic they see and may then determine other subscribers to the same topic. We also feel that since we are considering the case of web comics as an example and this sort of privacy does not exist in the current situation of viewing web comics via HTTP, this is not expected by likely subscribers.

Other work in this area has had different priorities from ours. Opychral and Prakash (2001) focus on ensuring that only authorised subscribers know the contents of a particular event. They propose an interesting group key caching scheme which is used to improve efficiency. They only encrypt the content during the last network hop from the publish/subscribe router to the subscriber. They do not endeavour to secure the content while it is being transferred from router to router and therefore they must trust all routers.

Raiciu and Rosenblum (2005) work on the problem of keeping the details of subscription filters and event notifications confidential (a secret from the routing nodes along the path) while maintaining the routers' ability to route the notifications to the correct subscribers. They assume that subscribers and publishers are honest and that secure channels are established. They also assume that the routers themselves are semi-honest and follow the protocol (for example, that the routers will forward an event if it matches and not just drop it).

We have chosen a different threat model and requirements, as discussed in the following section.

3 Threat model and requirements

We will consider two kinds of requirements for our publish/subscribe system: security requirements and performance requirements. Of these two, we prioritise security requirements but are still concerned with performance. We consider the additional work required for QUIP in our analysis of performance, as each publish/subscribe overlay network will have a different

baseline performance. We consider performance to be adequate provided that all types of transaction (subscription, unsubscription, publication of content, advertisement of new publications and unadvertisement of publications) occur within a few minutes. We do not require these to be instantaneous, but would like them to be as quick as possible. This level of performance is sufficient for many applications including a publish/subscribe network focused on web comics.

We propose a conservative threat model with as few assumptions as possible. An actual adversary may not have all the capabilities we propose, but we prefer to consider security against a worst case adversary. We consider an adversary to have the ability to add, delete and modify network traffic at both the underlying network layer and the publish/subscribe overlay layer. We also assume an adversary who may be a valid participant in the protocol or may collude with some other number of valid participants. We assume that the key server is trustworthy.

A summary of the security goals for the system are:

- To protect the content such that only authorised subscribers may read it
- To protect the payment information such that only the key server learns it
- To authenticate the source of messages from subscribers, publishers and the key server to one another
- To protect the integrity of messages in transit

Wang *et al.* (2002) provide a survey of security issues for publish/subscribe systems. They mention some security issues we have chosen not to address, such as subscription confidentiality and network availability. Although we have not addressed these issues directly, there is other work (Raiciu & Rosenblum 2005, Castro, Druschel, Ganesh, Rowstron & Wallach 2002) which does address these issues by making changes to the publish/subscribe overlay itself and could therefore still be used in conjunction with QUIP. Wang *et al.* also discuss content authentication and encryption, but assume that the only ways to achieve them involve either a *Public Key Infrastructure* or a set of keys shared between each publisher and subscriber. Using a PKI solution for content distribution is inefficient for a large number of subscribers since the content needs to be encrypted with each subscriber's individual key. Likewise, if we consider a system with a large number of shared keys we see that this would remove many of the benefits of the publish/subscribe system such as scalability and the separation of publishers from subscribers. Our traitor tracing approach using a key server avoids these problems.

In any restricted content system, it is possible for authorised subscribers to copy or release the content once they have validly decrypted it. It is difficult to see how this issue can be overcome and we do not attempt to resolve it in this paper. Our solution improves the situation by requiring the traitor to actively resend the decrypted content in an ongoing fashion or risk being caught for sharing the secret key. This increases the amount of work necessary for unauthorised copying and introduces some delay in the availability of unauthorised content.

4 Technical description of QUIP

We propose adding valuable security properties to existing publish/subscribe overlay services without

$S_A(x)$	x signed by participant A
$\{x\}_{K_A}$	Encryption of x with the key K_A which was chosen by participant A
x, y	Concatenation of x and y
$H(x)$	Cryptographic hash of x
n	A nonce (fresh unpredictable value)
ID_A	A unique identifier for participant A

Table 1: Explanation of notation used in equations

restricting the system to a particular overlay network. We do not assume that the overlay service provides any security properties. Eugster *et al.* (2003) discuss the main types of publish/subscribe system, *topic-based*, *type-based* and *content-based*. We believe QUIP could be used in conjunction with any of these types, however we have given examples for a *content-based* publish/subscribe system such as Siena (Carzaniga, Rosenblum & Wolf 1998, Carzaniga, Rosenblum & Wolf 2000).

There is a single trusted authority which will handle key management and payment called the key server. The key server separates subscribers from publishers. Each publisher sets the prices of their publications individually. If the publisher chooses to make their publication freely available, no keys are necessary so the key server is not involved in the transactions at all. If the publisher chooses to secure a publication, the authority makes sure that payment has occurred and provides the appropriate keys to the subscriber and publisher. QUIP uses a key management system described in Section 4.6. Unlike previous work such as Eventguard, when subscribers are added or removed in QUIP, only the publisher's key must be changed. Existing subscribers are not affected.

There are five major publish/subscribe operations which we want to secure. These are advertisement, unadvertisement, subscription, unsubscription and publication. We will discuss the protocol used for each of these operations in the sections below. We will also discuss an initialisation phase which happens only once as each new client joins the system.

We will describe the protocol we propose throughout Section 4 using the notation given in Table 1. Our protocol uses some basic cryptographic primitives including digital signatures, symmetric and asymmetric encryption, and cryptographic hash functions.

4.1 Initialisation

When an individual joins the service, they download a software client and are given a randomly generated identification number (ID) by the key server. The software client includes the public key of the key server, which is used to set up an initial secure tunnel. The key server will also generate a certificate linking this ID with a public key for each client. The certificates are used to authenticate signatures throughout the protocol. This enables us to require that messages be signed by both publishers and subscribers. We can then determine the ID of the participant who has originated a particular signed message. We have chosen to use a randomly generated ID to preserve some privacy related to the identity of clients. It is important to note that this privacy is only from other subscribers, publishers and attackers but not the key server. If the subscriber uses any paid service, the key server will have an entry in its database containing the subscriber's payment details linked with their ID .

In the initialisation phase, the client (designated as A in the equations) requests an ID and corresponding certificate from the key server (KS) as shown in

Equation 2. The initial request, a session key chosen by A and the public key of A are encrypted with the key server's public key which was downloaded with the QUIP client.

The key server generates a random ID and a certificate linking the public key provided by A to the new ID, encrypts them with the session key sent by A in the request and sends them back to A as shown in Equation 3.

$$A \rightarrow KS : \{ID_request, K_A, K_{A_pub}\}_{KS_pub} \quad (2)$$

$$KS \rightarrow A : \{ID_A, Cert_{ID_A}\}_{K_A} \quad (3)$$

4.2 Advertisement

Publishers must advertise their publications so that subscribers know what is available for subscription. This is done in two phases, the publisher must first inform the publish/subscribe overlay about the advertisement. If the publisher wishes to secure the publication, then the key server must also be informed. The publisher must send the key server the topic or title of the publication ($PubTitle$), the cost and the billing period (which is the same as the key change period). This message also includes the ID of the publisher and a fresh nonce as shown in Equation 4. These values are included so that these requests cannot be replayed by an attacker. The key server will verify that the signature on the message matches the ID contained in the message, that the nonce has not been used (by this ID) previously and that the $PubTitle$ is unique. If all these checks pass, the key server will respond as shown in Equation 5 confirming the request.

$$A_{publisher} \rightarrow KS : S_A(\text{Advertise}, PubTitle, Cost, BillingPeriod, ID_A, n) \quad (4)$$

$$KS \rightarrow A_{publisher} : S_{KS}(\text{AdvertiseAccept}, PubTitle, Cost, BillingPeriod, ID_A, n) \quad (5)$$

4.2.1 Unadvertisement

A publisher may choose to stop offering a publication. The revocation of an advertisement must be done both within the publish/subscribe system and the key server. The key server will then stop accepting subscriptions for the publication. The messages sent are identical to the ones above in Section 4.2 for advertisement except that the initial value changes from **Advertise** to **Unadvertise**.

4.3 Subscription

When a client wishes to subscribe to a new publication this is also done in two phases. First the subscription process within the publish/subscribe overlay is followed so that the client receives the content. Then the client will check if the content is encrypted or not. If the content is not encrypted, the client displays it. If the content is encrypted, the client contacts the key server to find out the subscription costs, as shown in Equation 6. A nonce is again included to ensure that the cost request and response are current. The key server responds giving the cost, the billing period and the start time of the next billing period, as shown in Equation 7. If the subscriber wishes to subscribe,

they will respond as shown in Equation 8. The key server will again check the signature and that all the details match before processing the request. The key server will request payment information (as described in Section 4.5) if it does not already have the details, or the details are rejected by the payment system. The key server will then check that the payment information is valid before providing the subscriber with the content decryption key which will begin to work at the start of the next billing cycle. This is shown in Equation 9.

$$A_{subs} \rightarrow KS : S_A(PubTitle, CostQuery, ID_A, n) \quad (6)$$

$$KS \rightarrow A_{subs} : S_{KS}(PubTitle, Cost, BillingPeriod, BillingStart, ID_A, n) \quad (7)$$

These cost query messages may be encrypted for extra privacy but we feel that in general it would not be worth the extra computation and complexity given that it would only hinder an attacker who is monitoring the conversation between the client and the key server but not the publish/subscribe overlay. An attacker who is monitoring the publish/subscribe overlay will know which publications a subscriber is receiving. It is possible for the subscriber to confound this sort of monitoring by subscribing to many publications without actually reading the content, but again, we feel these gains are probably not worth the effort.

$$A_{subs} \rightarrow KS : S_A(\{PubTitle, Subscribe, Cost, BillingPeriod, BillingStart, ID_A, n\}_{KS_pub}) \quad (8)$$

$$KS \rightarrow A_{subs} : S_{KS}(\{PubTitle, BillingStart, K_{PubTitle}, ID_A, n\}_{K_{ID_A_pub}}) \quad (9)$$

4.3.1 Unsubscription

When a subscriber no longer wishes to receive a publication, they must notify both the publish/subscribe overlay and the key server of their intentions. This is very straightforward. The subscriber uses the same message format as for subscribing as given in Equation 8 except that the keyword **Subscribe** is replaced with **Unsubscribe**.

4.4 Publication

Publication is the publish/subscribe operation which is performed most frequently. A publisher will encrypt and sign their publication with the current content key provided to them by the key server as shown in Equation 10. Also included is an *enabling block*, discussed in Section 4.6. The encrypted publication is then encapsulated inside the normal publish/subscribe publication format. The key server will generate a new content key for the publisher at each billing interval if the set of subscribers changes. It is important to note that the subscriber's content keys do not change.

$$A_{publisher} \rightarrow Publish/subscribe_overlay : S_A(\{K_{Content}\}_{Enabling_block}, Enabling_block, \{Publication\}_{K_{Content}}) \quad (10)$$

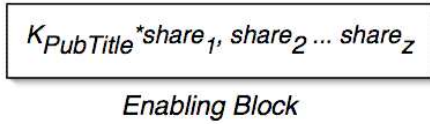


Figure 3: An illustration of the format of an enabling block.

4.5 Payment details

Occasionally it will be necessary for both subscribers and publishers to provide payment information to the key server. Since this information will change infrequently, it makes sense to send it only when necessary. The key server may request the payment details (as shown in Equation 11) or the publishers and subscribers may volunteer them at any time (as shown in Equation 12).

$$KS \rightarrow A_{client} : S_{KS}(\text{PaymentdetailsRequest}, ID_A, n) \quad (11)$$

$$A_{client} \rightarrow KS : S_A(\{\text{PaymentDetails}, ID_A, n\} K_{KS-pub}) \quad (12)$$

4.6 Key management and traitor tracing

The cryptography used has a significant impact on the performance of any secure system. We propose using a public key traitor tracing scheme designed by Tzeng and Tzeng (2001) which provides many advantages over symmetric and traditional asymmetric encryption schemes in this setting. There are two main advantages. First is the ability to revoke the keys of some subscribers without affecting the keys of the other subscribers. The second is the fact that each subscriber has a unique key which makes it easier to tell who has leaked a key if one is found in public or used by an unauthorised person. Further it is possible to discover who has leaked their keys even if they have combined their shares to create a new key. Once the traitor has been discovered the traitor's share can be revoked which will also disable the new key.

If the total number of subscribers is given by s , it is possible to revoke the keys of some subset of the total number of subscribers given by z . We must estimate s and generate that many shares of the key in advance. Encryption is done by creating an *enabling block* which contains z shares as shown in Figure 3. The enabling block is then used to encrypt the symmetric content key $K_{Content}$. This value is concatenated with the content encrypted with $K_{Content}$ as shown in Equation 10. The enabling block contains z shares (each subscriber key is a unique share), the shares contained in the enabling block are **not** able to decrypt the content key (and therefore not able to decrypt the content). When we wish to revoke a share, we simply include it in the enabling block when we encrypt. We fill in the rest of the z shares in the enabling block with unused shares. When the subscriber receives the publication they will first use their share of the publication key referred to as $K_{PubTitle}$ in Equation 9 to decrypt the content key ($K_{Content}$) and then use the content key to decrypt and read the publication.

By using a content key we can limit the extra computation necessary to use the public key traitor tracing scheme. The small (128 bit) content key is the only value we encrypt with the slower scheme. The computation time is also dependent on the number of

shares we can revoke (z) instead of the total number of subscribers s . By encrypting a constant length value (the content key) we know exactly how long it will take for each new publication. If the subscriber base has not changed, we can reuse the same *enabling block* for multiple publications. The actual content which may be of varying lengths is encrypted using the content key and a symmetric algorithm which will be significantly faster and may also be pre-computed once the content has been determined (before the billing period might have finished and the subscriber set is known).

The key server handles all key management for the system. This authority provides a single point of failure so we have kept the involvement of the authority to a minimum. The key server is involved only when the publisher chooses to require security for their publication. It may be necessary to issue a new key if the estimate of the number of subscribers s is too small and all the available shares have been used, or if the number of shares we wish to revoke is greater than z .

5 Performance analysis

Preliminary performance calculations based on the average computation time needed for each type of cryptographic operation show that a typical desktop PC is sufficient for use as the key server. We have shown the additional calculations required by QUIP from each of the participants for each publish/subscribe operation in Table 2 (additional relative to an unsecured publish/subscribe overlay). This is an improvement on the amount of computation required as compared to other secure additions to publish/subscribe systems mainly because the publisher only needs to encrypt each publication once. It is also an improvement on the number of messages that must be sent because any number of subscriber changes results in only one message from the key server to the publisher each billing period (a typical symmetric encryption based system would require a message to each subscriber and the publisher each billing period).

There is also a small amount of additional memory needed. In order to avoid replay attacks, the key server must keep a list of all the nonces used by a particular ID in subscription and advertisement requests.

6 Conclusion

QUIP addresses a number of important security issues in publish/subscribe systems. It allows subscribers to authenticate the messages they receive from publishers, and it permits publishers to restrict their messages to authorised subscribers and to add and remove subscribers efficiently. It represents a significant improvement over previous work in securing publish/subscribe networks because it properly addresses the issue of key management and therefore does not need to make strong assumptions about the security of the underlying infrastructure. The traitor-tracing approach makes addition and revocation of subscribers much more efficient than in previous work. Publishers who are happy for their content to be freely distributed do not need to do any extra work even if the publish/subscribe system offers QUIP.

However, any security protocol makes some assumptions about the power of the adversary. Perhaps the most serious in our paper is the assumption that the key server cannot be subverted. This represents a single point of failure. An interesting open question is whether Tzeng and Tzeng's traitor-tracing scheme

Participant	Operation	Extra computation
Client	Initialise	Asymmetric encrypt, Generate public/private key pair
Key Server	Initialise	Generate random ID, Create certificate, Symmetric encrypt
Subscriber	Subscribe	Signature
Key Server	Subscribe	Signature check, Update subscriber set
Subscriber	Unsubscribe	Signature
Key Server	Unsubscribe	Signature check, Update subscriber set
Publisher	Advertise	Signature
Key Server	Advertise	Create new public traitor tracing key shares
Publisher	Unadvertise	Signature
Key Server	Unadvertise	Signature check
Publisher	Publish	Signature, Public traitor tracing encrypt content key, Symmetric encrypt content
Subscriber	Publish	Signature check, Public traitor tracing decrypt content key, Symmetric decrypt content

Table 2: Additional computation required by QUIP

can be extended to allow the key server to be distributed. Ideally, we could both improve security and reduce the load on each server by having several key servers, of which each subscriber needed to contact a subset. Then the adversary would have to subvert several servers.

A related but more difficult issue is the auditability of the key server, as mentioned by Wang *et al* (2002). A publisher who is suspicious of being underpaid has no way of testing the key server's honesty. The server can simply claim that there are fewer paying subscribers than there really are. Unfortunately it is difficult to see how the publisher could test this except in some out-of-band way. It seems unreasonable to assume that a publisher can observe communications to or from the key server. Another issue equally difficult to resolve is the problem of cheating publishers, who collect their money but fail to deliver appropriate content when promised. For example, it seems impossible for the system to detect when a cartoon has been published before, or is too dark or blurry to read. Perhaps a reputation system would mitigate this problem.

Depending on the content being delivered, different applications of publish/subscribe systems have different security requirements. We have addressed one such set of requirements in this paper, and we expect our solution to be consistent with other protocols that provide subscription confidentiality or guarantee message delivery.

References

- Carzaniga, A., Rosenblum, D. S. & Wolf, A. L. (1998), Design of a scalable event notification service: Interface and architecture, Technical Report CU-CS-863-98, Department of Computer Science, University of Colorado.
- Carzaniga, A., Rosenblum, D. S. & Wolf, A. L. (2000), Achieving scalability and expressiveness in an internet-scale event notification service, *in* 'Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing', Portland, Oregon, pp. 219–227.
- Castro, M., Druschel, P., Ganesh, A., Rowstron, A. & Wallach, D. S. (2002), Secure routing for struc-

tured peer-to-peer overlay networks, *in* 'Proceedings of the 5th Usenix Symposium on Operating Systems Design and Implementation (OSDI 2002)'.

Eugster, P. T., Felber, P. A., Guerraoui, R. & Kermarrec, A.-M. (2003), 'The many faces of publish/subscribe', *ACM Computing Surveys* **35**(2), 114–131.

Opyrchal, L. & Prakash, A. (2001), Secure distribution of events in content-based publish/subscribe systems, *in* 'Proceedings of the 10th USENIX Security Symposium'.

Raiciu, C. & Rosenblum, D. S. (2005), Enabling confidentiality in content-based publish/subscribe infrastructures, Technical report, University College London:w Technical Report RN/05/30.

Srivatsa, M. & Liu, L. (2005), Securing publish/subscribe overlay services with eventguard, *in* 'Proceedings of the 12th ACM conference on Computer and communications security'.

Tzeng, W.-G. & Tzeng, Z.-J. (2001), A public-key traitor tracing scheme with revocation using dynamic shares, *in* 'Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography: Public Key Cryptography', Vol. 1992 of *Lecture Notes In Computer Science*, Springer-Verlag, London, UK, pp. 207–224.

Wang, C., Carzaniga, A., Evans, D. & Wolf, A. L. (2002), Security issues and requirements for Internet-scale publish/subscribe systems, *in* 'Proceedings of the Thirty-Fifth Annual Hawaii International Conference on System Sciences', Big Island, Hawaii.

Enhancing Data Locality in a Fully Decentralised P2P Cycle Stealing Framework

Richard Mason

Wayne Kelly

Programming Languages and Systems Research Group
Queensland University of Technology,
Brisbane, Queensland 4001,
Email: r.mason,w.kelly@qut.edu.au

Abstract

Peer-to-peer (P2P) networks such as Gnutella and BitTorrent have revolutionised Internet based applications. P2P approaches provide a number of benefits, however most cycle stealing projects, such as SETI@home, have concentrated on centralised methods which still require massive amounts of concentrated network bandwidth in order to scale. More recent P2P research has developed the concept of distributed hash table (DHT) P2P overlays. These overlays provide efficient and guaranteed message delivery unlike earlier P2P networks which relied on large scale replication to probabilistically find data. Our G2:P2P framework makes use of a DHT overlay to provide a fully decentralised P2P cycle stealing system. Its distributed object programming model allows direct communication between objects and it remains reliable even as the set of peer nodes changes.

In this paper we describe extensions to G2:P2P which allow us to optimise object distribution for locality. The importance of optimising data locality is well understood and has received extensive research, however, in the context of cycle-stealing systems and more generally DHT based P2P networks it is completely unexplored. Whilst our work is motivated by parallel programming, it is generic in nature and may have applicability to other DHT applications.

Keywords: Peer-to-Peer, Cycle stealing, locality, distributed hash table

1 Introduction

Peer-to-peer networks have become popular of late for a range of applications, including the well known file sharing systems such as Gnutella (Kan 2001) and BitTorrent (Cohen 2003). Peer-to-peer in general refers to distributed systems where resources are obtained from an ad hoc collection of client computers (as opposed to dedicated central servers). Typically the set of peer nodes participating in the network changes over time. Early P2P systems such as Napster (Shirky 2001) used an underlying client-server architecture to facilitate communication between peers. Later, fully decentralised (pure) P2P networks were developed to eliminate the impact of a single point of failure and to make it cheaper to scale. These early pure P2P networks were inefficient, requiring a great deal of network bandwidth which prevented their use on lower bandwidth connections (Hong 2001). Their

design also meant that data present in “distant” parts of the network may not always be found.

Pure P2P research has since centred around the idea of a distributed hash table (DHT) (Rowstron & Druschel 2001, Zhao, Huang, Stribling, Rhea, Joseph & Kubiawicz January 2004). A hash function is used to map objects to be stored in the P2P network to the peer node that should hold that object. The peer node whose randomly assigned identifier is “closest” to an object’s hash value will always hold that data. As the set of peers that make up the P2P network changes over time, the ownership of data changes so as to maintain the above invariant. It is this invariant that allows messages to be efficiently routed (average $O(\log N)$ hops) to arbitrary objects without needing to update object references and without needing to resort to forwarding schemes.

While DHTs have been used for a number of applications (Rowstron, Kermarrec, Castro & Druschel 2001, Rhea, Eaton, Geels, Weatherspoon, Zhao & Kubiawicz March 2003), we believe our G2:P2P framework (Mason & Kelly 2003) is the first parallel computing/cycle-stealing framework to exploit this approach. Most volunteer/cycle-stealing systems are based on client-server (Kelly, Roe & Sumitomo June 2002) or hybrid (Neary, Brydon, Kmiec, Rollins & Cappello 2000) topologies. This typically limits them to simple task based embarrassingly parallel applications. Our DHT based cycle-stealing framework supports a distributed object programming model where remote method invocations can be made between arbitrary peers in the network and remains reliable even as the set of peer nodes that make up the network changes over time.

In this paper we describe extensions to our parallel computing framework that allow us to optimise for locality. The importance of optimising for locality in parallel programs is well understood and there has been extensive work in this area. However, in the context of cycle-stealing systems and more generally DHT based peer-to-peer systems this topic has been completely unexplored. As it turns out, there is room for movement within the DHT concept that allows us to achieve our new goals related to locality while preserving all of the original benefits of DHTs that have made them so popular. Whilst our work is motivated by parallel programming, it is entirely likely that the locality ideas proposed here will have wider applicability to other DHT applications.

Section 2 provides an overview of our G2:P2P framework. It includes details on the underlying Pstry network as well as the G2:P2P object model and programming interface. In Section 3 we describe four schemes for improving object locality. The schemes build upon one another to provide increasing performance improvements. Section 4 demonstrates the effect of these optimisations on communication and load balancing performance. In Section 5 we discuss

other related work before outlining future directions and concluding in Section 6.

2 G2:P2P Overview

G2:P2P is a fully decentralised (pure) P2P system for executing parallel applications on volunteer machines. The system provides a distributed object based programming model.

A major difference between G2:P2P and existing cycle-stealing frameworks is the ability to perform direct communication between tasks while they are executing. Previous systems have, at best, only provided communication using a server as an intermediary. By taking advantage of direct peer-to-peer communication, G2:P2P significantly increases the range of applications that may take advantage of cycle-stealing.

2.1 Pastry

A number of DHT approaches (Zhao et al. January 2004, Ratnasamy, Francis, Handley, Karp & Shenker 2000, Stoica, Morris, Karger, Kaashoek & Balakrishnan 2001) have been developed with very similar characteristics. Our framework is based on the Pastry (Rowstron & Druschel 2001) DHT system. Pastry networks consist of a set of machines (nodes) which are each assigned a unique k -bit key termed a NodeID. The assignment of NodeID's to nodes must be done in a fully decentralised manner but in such a way that the NodeIDs are approximately evenly distributed across the address space (so as to achieve approximately even load balance). The simplest approach is to assign them in a pseudo-random fashion (either by generating random GUIDs or by hashing their IP addresses). The address space generated by these NodeIDs is circular; i.e. for an example 8-bit address space address, 0xFF would be adjacent to 0x00.

Messages may be addressed to any NodeID in the address space, regardless of whether an actual node exists with that ID or not, and will be delivered to the node whose ID is closest to the target. Pastry's routing scheme will deliver messages in $O(\log N)$ hops using a scheme reminiscent of hypercube routing but with dynamically changing nodes.

Nodes have two notions of locality: physical locality and virtual locality. Physical locality represents closeness with respect to the physical network (the closer the physical locality – the shorter the message latency). When lacking any better physical locality data we generally assume that the numeric difference between two IP addresses is a rough approximation to their physical locality (i.e. two nodes on the same subnet will generally be closer than two nodes on different subnets, etc). Virtual locality is measured by the difference between NodeIDs.

Nodes maintain information to facilitate the routing of messages. Each node remembers a small set of the nodes (size $O(1)$) that are closest to it physically (termed its *neighbourhood set*), and closest to it virtually (termed its *leaf set*). The primary routing mechanism is a routing table containing $O(\log N)$ entries that allows each node to forward any incoming message onto a node that is at least one significant bit closer to its final destination.

2.2 Object Addressing and Programming Model

G2:P2P applications create objects which are distributed on a network of volunteers organised in a Pastry network. Communication between these objects is performed through remote method calls (as

in .NET Remoting (Obermeyer & Hawkins 2001) or Java RMI (Sun Microsystems 2004)). However, when using G2:P2P the application programmer does not need to specify a specific server to host a particular class of remote objects, but rather simply indicates that they should be hosted somewhere within the collection of volunteer machines. In fact the volunteer that hosts a given remote object may change over time as volunteers come and go from the network.

When an object is created, it is assigned an ObjectID which has the same form as a Pastry NodeID. A construction message is then sent to that ID, informing the node with the closest ID (i.e. the node that receives the message) that it should create a remote object instance. The only information required to refer to and communicate with the object is its ObjectID. If another node joins the network and is assigned a NodeID which is closer to the object then the object should be migrated to the new node. The G2:P2P framework automatically performs these migrations as well as performing logging and check pointing to maximise reliability.

3 Extensions for Improving Communication Locality

Objects are assigned an ObjectID when they are created. It is not possible to change an object's ObjectID at a later stage as references to that object may have spread throughout the network and they would need to be updated in a globally synchronised manner. We still however, have some freedom that allows us to optimise for locality. The following four subsections describe four separate schemes that we have developed to optimise different kinds of locality.

3.1 ObjectID Assignment

Our first extension is designed to increase the likelihood that communicating objects are hosted on the same node. Previously our approach to assigning ObjectIDs was to generate them randomly (similar to the allocation of NodeIDs). This was done so as to approximately evenly spread the ObjectIDs over the address space so as to achieve approximately even load balance. However, unlike NodeIDs, ObjectIDs do not need to be generated de-centrally – a single client or node often activates a collection of related remote objects at a time. Further, these remote objects typically do not exhibit purely random communication behaviour (patterns such as “nearest neighbour” are very common).

By allocating ObjectIDs in a more intentional manner we can optimise for communication locality while preserving load balance. Our new approach is to allocate ObjectIDs to a collection of related objects so that they are uniformly rather than randomly distributed over the address space. We do this in such a way that objects that communicate regularly are closer together with respect to their ObjectIDs than objects which do not regularly communicate. For this optimisation to be beneficial we assume that the number of remote objects created will be much larger than the number of volunteers, so by assigning similar ObjectIDs to objects that communicate regularly, we ensure that in most cases they will be assigned to the same physical host. The set of volunteer nodes participating in the network may change over time and therefore so will the mapping of remote objects to volunteer nodes, but the locality properties will be preserved.

Our notion of closeness is based on ObjectIDs and is therefore 1 dimensional (more precisely a 1D-ring). Other communication patterns between objects must

therefore be “folded” by the programmer onto this 1D space.

The API for allocating ObjectIDs in this way basically tells the system to start uniformly assigning ObjectIDs to any remote object activations that are about to be performed. The StartLayout method needs to know the number, N , of objects that will be created in this group, so that they will be properly spaced to collectively cover the entire address space. Following this call, the next N remote object activations will automatically be assigned an ObjectID using this scheme.

```
Island[] islands = new
Island[numIslands];
G2P2PChannel.Current.StartLayout(numIslands);
for (int i = 0; i < numIslands; i++)
    islands[i] = new Island(i);
```

The same pattern can be repeated later if another set of remote objects need to be created that are unrelated to the original set (with respect to communication pattern). Later allocations will use a different random offset so as to not collide with the original uniformly generated IDs.

Section 4.1 presents the results of the new ObjectID assignment process on communication efficiency and load balancing.

3.2 Object Groups

The optimisation in Section 3.1 increases the chances that two objects that communicate often will be located on the same machine. It cannot however guarantee that they will always be hosted on the same machine. The optimisation described in this section is designed for situations in which a set of objects communicate so frequently with one another that they should always be collocated on the same host. The way we do this is by adding extra bits to our ObjectIDs beyond the length of our NodeIDs. We can think of this as turning ObjectID into decimal numbers rather than integers. Objects which share the same integer part will always be mapped to the same node. The actual physical node may change over time as nodes come and go, but the group of objects will always be collocated. Note that this locality comes at the cost of load balance and therefore parallelism. If a group of objects are assigned the same integer part, they will always map to a single machine, even if there are a large number of other nodes in the network which are completely unused.

An alternative to this optimisation is to encapsulate these objects inside a single remote object container that forwards messages to them. The advantage of the approach we have described is that each of the objects in the group remain individually addressable by remote clients. Whether or not a set of objects should always be collocated is an performance optimisation which ideally should be kept separate from the application logic and the abstractions used.

3.3 Node Movement

The assignment of ObjectIDs and NodeIDs described so far will lead to approximately the same number of objects being allocated to each node. In some cases however, particularly with smaller networks, this balance will not be achieved.

As we discussed earlier, it is extremely problematic to change an object’s ObjectID after it is initially assigned as references to that object may have spread throughout the entire network. It is, however, possible for a node’s NodeID to change at a later time. A simple way to explain why this is possible is to view

the process as equivalent to a volunteer node leaving the network and then immediately rejoining (with a new NodeID). So, clearly it is possible, and with a little ingenuity we can develop a process that is much more efficient than the naïve implementation hinted at above.

But why would we want to change a node’s NodeID? What we generally want to do is reassign NodeIDs so that each node gets a more even number of objects. To do this perfectly we would need global knowledge and even if we had such global knowledge, what might be optimal one minute might not be the next as nodes and objects come and go over time. As with all decentralised systems we must attempt to achieve close to global optima through local acts.

At the local level a node’s goal is to try to evenly space itself between its two neighbours. Since each node is trying to do this independently it may take a large number of small adjustments to get to a steady state. To minimise the total number of adjustments nodes may make use of the extra information they contain in their leaf set. Instead of simply placing itself halfway between its two immediate neighbours, it measures the distances to all of its known neighbours and attempts to balance them.

It is important that these NodeID changes are moving incrementally towards a global optima. If the changes are too dramatic the system may not converge, in particular, we do not want nodes to move so far that they move past other nodes and thereby change their relative order. By keeping nodes relative order stable, their leaf set will remain constant. This guarantees that the Pastry routing mechanism will continue to function correctly. We therefore use the following formulas to calculate incremental NodeID changes.

$$N' = N + S \frac{(W_{anti} - W_{clock})}{2}$$

$$W_i = \sum_{n \in LS_i} 1 - \frac{(N - n_{pos} n_{weight})}{S_{leafset}}$$

$$N = \text{Node position}$$

$$S = \text{Size of network}$$

$$S_{leafset} = \text{size of leaf set}$$

Where W_i indicates the weight of the respective half of the leaf set and LS_i represents the set of nodes in each half of the leaf set. Essentially W_{clock} and W_{anti} calculate the force expressed on the node by the nodes by the clockwise and anti-clockwise half leaf set. Nodes that are further away express less force than nearby nodes. The new position is calculated to equalise those two forces.

We also include a weighting factor (n_{weight}) on each node. This allows us to take in to account peer nodes that have greater processing power (e.g. a cluster computer rather than a PC). Such nodes will be responsible for a greater portion of the address space and hence will host more objects.

Since it is imperative that we maintain the relative order of the nodes we further restrict node movement so that any single move may not travel more than half the distance to the next node. Without this restriction nodes may “cross over” each other as they independently calculate their moves. This restriction may slow down the progress towards the global optima, however, once the network is well dispersed node movements are generally small anyway and this restriction is rarely encountered.

Once a node calculates a new position it informs all of the members of its leaf set. Additionally it must now monitor incoming communications to detect other nodes that have references to it (such as references in their routing table). When these are detected the node can respond with its new id. Since

most movements will be relatively small, messages sent via a routing table will usually still send the message closer to its target, and even if it does not, routing can continue with a temporary disadvantage of some extra network hops.

The node's routing state must be updated to reflect the new NodeID. The leaf set is not affected because the nodes with adjacent NodeIDs are guaranteed not to change. Similarly, the neighbourhood set is unchanged since it is only related to physical locality, not the NodeID. The amount of the routing table affected is directly related to the distance moved by the node. Generally portions of the routing table will stay valid, while later sections will need to be repopulated.

When a node joins there is a significant opportunity to optimise its location before it even advertises itself to other nodes. As part of our node movement technique we extend the joining process to reduce the number of movements a node must make before the network stabilises. The standard Pastry join procedure starts with the new node routing a special join message to its prospective NodeID. This is received by the node whose ID is closest to the new position who then replies with confirmation of the IDs acceptance and with some initial data to start populating the node's routing state. With minimal changes we can alter the processing of join messages so that the receiving node can reply with a different NodeID for the node to use during joining. This new ID can be selected so that it already has a balanced leaf set, reducing the need for further adjustments directly after joining.

Section 4.2 presents the results of moving node's IDs on communication efficiency and load balancing.

3.4 Physically Related Nodes

The optimisation in Section 3.1 increases the chances that two objects that communicate often will be located on the same machine. There will however usually be situations where objects that communicate often are located on different machines. If they cannot be on the same machine then we would prefer that they are on machines which are physically close. To achieve this we continue to assign ObjectIDs as described in Section 3.1 but change NodeIDs so that nodes with similar NodeIDs will be physically close to one another. If two objects communicate frequently then they will be assigned similar ObjectIDs. If we are lucky this will mean they will be hosted on the same node. If not, then it they will be hosted on nodes that have similar NodeIDs which implies they are physically close.

Our basic approach is to use IP addresses (or some other measure of physical locality) as our NodeIDs. This ensures that nodes with similar NodeIDs will tend to be physically close. The problem with this approach is that the resulting NodeIDs will not be evenly spread over the address space. Thankfully, the optimisation described in Section 3.3 can correct that situation.

However, using the node movement technique from Section 3.3 complicates the joining process. As nodes adjust their IDs, they may find themselves moving significantly away from the ID that was generated from their IP address. This means that we need to alter the way join messages are routed so that new nodes can still be placed adjacent to nodes with similar IP addresses. To do this we introduce two NodeIDs – the initial NodeID which was generated directly from the node's IP address and the current NodeID which is being used by the node. While normal communication messages use the current NodeID

for routing, join messages will be routed to the node whose initial ID is closest.

Unfortunately the routing table maintained by each node is designed for standard routing and hence can not be used for routing to initial IDs. We can however rely on the order of current NodeIDs being the same as the initial NodeIDs. This means we can use the leaf sets to route messages, albeit with a worse case of $N/2$ network hops. In practice, joining is generally performed by contacting a physically close node and using it to initiate the join message. This means that regardless of how far node IDs move during execution, join messages will always be initiated reasonably close to their final target.

There is one disadvantage of applying this optimisation. In a normal Pastry network, nodes in a particular physical locality are likely to be widely spread throughout the network (with respect to their NodeIDs). So, if some fault in that physical locality occurs (such as a local power loss or a local network going down) then loss of nodes will be felt in a more dispersed fashion across the network rather than in a single large cluster of nodes. Pastry networks are designed to be able to recover from individual nodes disappearing, provided that other nodes in its leaf set remain. So, by changing this aspect of the Pastry network we decrease its ability to recover from local faults. This is obviously a trade-off that must be made between efficiency and reliability. Our previous paper (Mason & Kelly 2005) describes additional techniques that we have developed to enhance reliability.

4 Results

The optimisations presented in Sections 3.1 and 3.3 have been implemented in the actual G2:P2P system as well as a simulator. Tests have been performed in both systems, with the simulator providing the ability to test larger networks.

4.1 Simulator

The simulator generates a Pastry network and simulates a set of objects communicating periodically in a 1D nearest neighbour configuration.

We use two metrics to analyse the effectiveness of our enhancements. The first is the *communication efficiency*. This is the ratio of messages sent between objects to the number of network hops used by those messages. An efficiency of one indicates that each message required a single hop across the network while higher values indicate that some messages were delivered without requiring the network.

The simulator has not taken into account the physical communication costs between different nodes. Pastry already accounts for physical locality in its selection of which nodes to use for network hops. The work in Sections 3.1 and 3.3 is mainly concerned with minimising the number of hops required.

Our second metric, the *load balance rating*, measures how well balanced the objects are across the network. This is provided by calculating the standard deviation of the number of objects on each node, lower values indicating that each node was carrying approximately the same load.

4.1.1 Enhanced ObjectID Results

Our first test shows the effect of our enhanced ObjectID assignment. The even spacing of ObjectIDs significantly improves the communication efficiency of the network especially as the number of objects outstrips the number of nodes. Load balancing is also

mildly improved, though that effect declines as more nodes are added to the network.

Figures 1 & 2 show the results of activating the ObjectID assignment optimisations. Dotted lines indicate tests without the optimisation while solid lines indicate tests with the optimisation activated. Each line indicates a separate test with the label indicating the number of volunteer nodes involved.

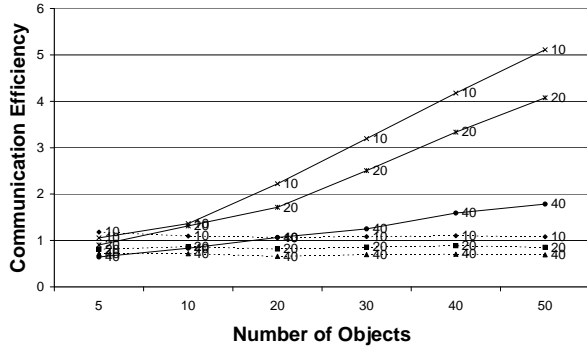


Figure 1: ObjectID Adjustment Communication Efficiency

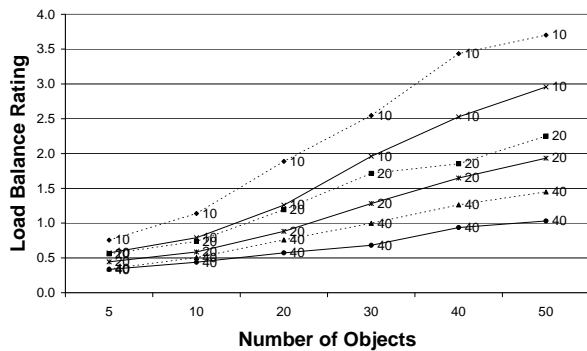


Figure 2: ObjectID Adjustment Load Balance Rating

4.1.2 Movement Results

The second test shows the effect of adding the node movement extensions to the new ObjectID assignment method. As expected we find that node movement significantly improves the load balancing, though the effect decreases as more nodes are added to the system. Communication efficiency on the other hand decreases slightly when node movement is added. This is directly related to the load balancing, since as the objects spread amongst all of the nodes, less communication is performed internally within nodes.

Figures 3 & 4 show the results of activating the node movement optimisations. Dotted lines indicate tests without node movement while solid lines indicate tests with the optimisations activated. As before, each line indicates a separate test with the label indicating the number of volunteer nodes involved.

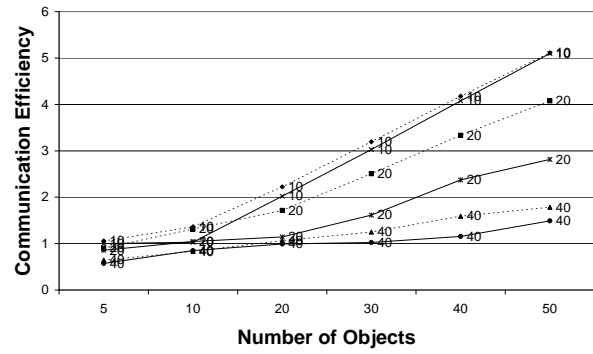


Figure 3: Node Movement Communication Efficiency

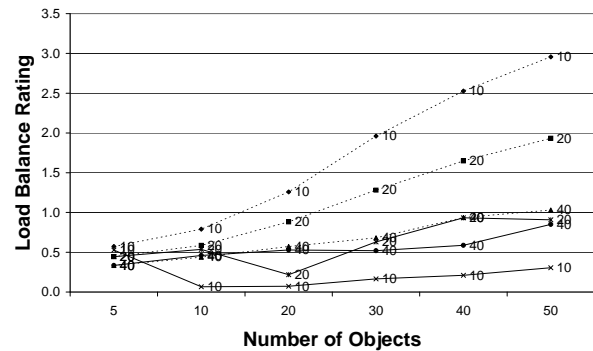


Figure 4: Node Movement Load Balance Rating

4.2 Actual System

Tests were also performed using an actual G2:P2P network running on eight 2.6 GHz Pentium 4 machines connected through a Gigabit Ethernet network. The test application creates a set of objects which each run a series of calculations separated by synchronisation stages where the object sends a message to its two nearest neighbours and waits for a similar message from them. The tests can be varied by altering the number of objects in the application, the size of the synchronisation messages and the length of the processing step.

Each test was performed 3 times – once without any optimisations applied, once with the enhanced ObjectID assignment and finally with both the ObjectID assignment and the node movement enhancements activated. Each test consisted of 10 runs with the same parameters from which an average execution time was gathered. Tests were repeated for differing numbers of objects and differing processing step lengths. Testing was also performed with differing message sizes, however they did not display significant differences and have not been included in this paper. All results shown were from tests using 1KB messages.

Two processing lengths were used. The first taking approximately 0.12 seconds per object to complete on an unburdened test machines and the second taking approximately 0.39 seconds. Figure 5 shows the results of the tests run at the shorter processing length, while Figure 6 shows the results of the longer processing tests.

The tests show that the optimisations presented

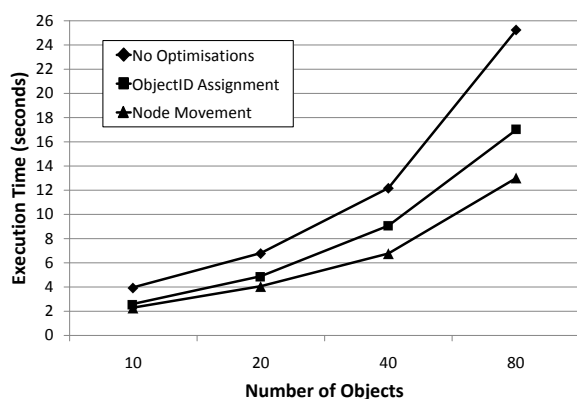


Figure 5: Average time taken to run short processing test

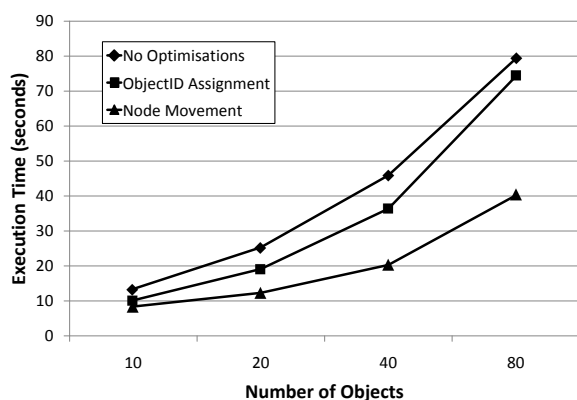


Figure 6: Average time taken to run long processing test

greatly improved the performance of the system with run times effectively halved when both optimisations are applied. The tests also support the results gathered on the simulator. In Figure 5 the ObjectID assignment provide good speedup over the unoptimised test, however this speedup is not as significant in the second test set. In this set, where the load is increased, the ObjectID assignment is less significant as overloaded volunteers continue to slow down the entire application. The load balancing benefits of the node movement optimisation overcomes this issue and provides the majority of speedup in this case.

5 Related Work

Current cycle-stealing systems are predominantly client-server based (Kelly et al. June 2002, Anderson November 8, 2004, Cappello, Djilali, Fedak, Herault, Magniette, Néri & Lodygensky 2005) or use a hierarchical layout to improve scalability (Cappello & Coakley 2005, Baratloo, Karaul, Kedem & Wyckoff 1996). Applications making use of these frameworks are generally limited to embarrassingly parallel approaches, however, more recently work has been performed to allow a other application models such as branch and bound (Neary et al. 2000) and continuations (Kelly, Roe & Sumitomo 2003). G2:P2P's

direct inter-object communication is unavailable, and difficult to imitate, using the client-server approach of these systems.

Condor (Tannenbaum, Wright, Miller & Livny 2001) provides direct communication between cycle-stealing processes using messaging APIs like MPI and PVM, however this feature requires a highly reliable environment. Similarly, Knitting Factory (Baratloo, M., Karl & Kedem 1997) uses Java RMI for communication, but is unable to cope with volunteers leaving the system. G2:P2P provides direct communication and is reliable across volunteer arrivals, departures and failures.

Some other attempts have been made in creating decentralised cycle-stealing applications (Gupta & Somani 2004, Awan, Ferreira, Jagannathan & Grama February 2006), but this research concentrates on the economic aspects of cycle-stealing and does not address how pure P2P features such as direct communication can extend the programming model of cycle-stealing applications. The DREAM project (Arenas, Collet, A. E. Eiben, Merelo, Paechter, Preuß & Schoenauer 2002) provides an innovative approach for doing pure P2P cycle-stealing but primarily concentrates solely on the field of evolutionary algorithms.

6 Future Work & Conclusions

G2:P2P's direct object-to-object communication ability is unique amongst cycle stealing projects. The extensions explored in this paper effectively enhance this communication feature, significantly improving its performance for common communication patterns. Additionally, the alterations made to the Pastry layer provide some generic enhancements which may be beneficial to other applications using distributed hash table P2P overlays.

There is still scope for further improvements to object locality. In particular, support for a wider range of communication patterns. Currently nearest neighbour communication is well supported. This is the obvious starting point as it maps well to the Pastry address space, however methods for supporting other topologies such as mesh or tree based communication may also be able to be supported.

A secondary benefit of this locality work has been a substantial improvement to the load balancing of G2:P2P, particularly for smaller networks. There is scope for further work in this area taking into account the variety in processing power of volunteers and the workload incurred by different objects. Like the enhancements presented here, adjusting allocation for different loads is difficult, though not unachievable, in fully decentralised networks.

7 References

References

- Anderson, D. P. (November 8, 2004), Boinc: A system for public-resource computing and storage, in '5th IEEE/ACM International Workshop on Grid Computing, Pittsburgh, USA'.
- Arenas, M. G., Collet, P., A. E. Eiben, M. J., Merelo, J. J., Paechter, B., Preuß, M. & Schoenauer, M. (2002), 'A framework for distributed evolutionary algorithms', *Lecture Notes in Computer Science* **2439**, 665–675.
- Awan, A., Ferreira, R. A., Jagannathan, S. & Grama, A. (February 2006), 'Unstructured peer-to-peer networks for sharing processor cycles', *Parallel Computing* **32**(2), 115–135.

- Baratloo, A., Karaul, M., Kedem, Z. & Wyckoff, P. (1996), Charlotte: Metacomputing on the web, in 'Proc. of the 9th Int'l Conf. on Parallel and Distributed Computing Systems (PDCS-96)'.
- Baratloo, A., M., K., Karl, H. & Kedem, Z. M. (1997), Knittingfactory: An infrastructure for distributed web applications, Technical Report TR1997-748, New York University.
- Cappello, F., Djilali, S., Fedak, G., Herault, T., Magniette, F., Néri, V. & Lodygensky, O. (2005), 'Computing on large-scale distributed systems: Xtrem web architecture, programming models, security, tests and convergence with grid', *Future Gener. Comput. Syst.* **21**(3), 417–437.
- Cappello, P. & Coakley, C. J. (2005), Jicos: A Java-Centric Networking Computing Service, in S. Zheng, ed., 'Proc. 17th IASTED Int. Conf. Parallel and Distributed Computing and Systems', pp. 510 – 515.
- Cohen, B. (2003), 'Incentives build robustness in bittorrent', <http://www.bittorrent.com/bittorrentecon.pdf>.
- Gupta, R. & Somani, A. (2004), Compup2p: An architecture for sharing of computing resources in peer-to-peer networks with selfish nodes, in 'Proceedings of the Second Workshop on the Economics of Peer-to-Peer Systems, Cambridge, MA'.
- Hong, T. (2001), *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, O'Reilly & Associates, Inc, chapter Performance.
- Kan, G. (2001), *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, O'Reilly & Associates, Inc, chapter Gnutella.
- Kelly, W., Roe, P. & Sumitomo, J. (2003), An enhanced programming model for internet based cycle stealing, in H. R. Arabnia & Y. Mun, eds, 'PDPTA', Vol. 4, CSREA Press, pp. 1649–1655.
- Kelly, W., Roe, P. & Sumitomo, J. (June 2002), G2: A grid middleware for cycle donation using .net, in '2002 International Conference on Parallel and Distributed Processing Techniques and Applications'.
- Mason, R. & Kelly, W. (2003), Peer-to-peer cycle sharing via .net remoting, in 'Proceedings of the Ninth Australian World Wide Web Conference (AusWeb03)'.
- Mason, R. & Kelly, W. (2005), G2-p2p: A fully decentralised fault-tolerant cycle-stealing framework, in 'Proceedings of the Australasian Workshop on Grid Computing and e-Research (AusGrid05), Newcastle, Australia'.
- Neary, M. O., Brydon, S. P., Kmiec, P., Rollins, S. & Cappello, P. (2000), 'Javelin++: scalability issues in global computing', *Concurrency: Practice and Experience* **12**(8), 727–753.
- Obermeyer, P. & Hawkins, J. (2001), 'Microsoft .net remoting: A technical overview', <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/hawkremoting.asp>.
- Ratnasamy, S., Francis, P., Handley, M., Karp, R. & Shenker, S. (2000), A scalable content addressable network, Technical Report TR-00-010, Berkeley, CA.
- Rhea, S., Eaton, P., Geels, D., Weatherspoon, H., Zhao, B. & Kubiawicz, J. (March 2003), Pond: the oceanstore prototype, in 'Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST'03)'.
- Rowstron, A. & Druschel, P. (2001), 'Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems', *Lecture Notes in Computer Science* **2218**, 329–350.
- Rowstron, A. I. T., Kermarrec, A.-M., Castro, M. & Druschel, P. (2001), SCRIBE: The design of a large-scale event notification infrastructure, in 'Networked Group Communication', pp. 30–43.
- Shirky, C. (2001), *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, O'Reilly & Associates, Inc, chapter Listening to Napster, pp. 21–37.
- Stoica, I., Morris, R., Karger, D., Kaashoek, M. F. & Balakrishnan, H. (2001), Chord: A scalable peer-to-peer lookup service for internet applications, in 'Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications', ACM Press, pp. 149–160.
- Sun Microsystems (2004), 'Java remote method invocation - distributed computing for java', <http://java.sun.com/products/jdk/rmi/reference/whitepapers/javarmi.html>.
- Tannenbaum, T., Wright, D., Miller, K. & Livny, M. (2001), Condor – a distributed job scheduler, in T. Sterling, ed., 'Beowulf Cluster Computing with Linux', MIT Press.
- Zhao, B. Y., Huang, L., Stribling, J., Rhea, S. C., Joseph, A. D. & Kubiawicz, J. D. (January 2004), 'Tapestry: A resilient global-scale overlay for service deployment', *IEEE Journal on Selected Areas in Communications* **22**(1).

Hybrid Mesh Ad-hoc On-demand Distance Vector Routing Protocol

Asad Amir Pirzada, Marius Portmann* and Jadwiga Indulska*

Queensland Research Laboratory,
National ICT Australia Limited,
Brisbane, QLD 4000, Australia.

{asad.pirzada,marius.portmann,jadwiga.indulska}@nicta.com.au

Abstract

Wireless Mesh Networks (WMNs) have recently gained increasing attention and have emerged as a technology with great potential for a wide range of applications. WMNs can be considered as a superset of traditional mobile ad-hoc networks (MANETs), where the network is comprised of mobile client devices (MESH_CLIENTs). In addition to MESH_CLIENTs, a WMN can also contain relatively static devices called mesh routers (MESH_ROUTERS). Such *hybrid WMNs* are characterized by a high level of heterogeneity, since static MESH_ROUTERS are typically much less resource constrained than mobile MESH_CLIENTs, and are also often equipped with multiple radio interfaces. Traditional ad-hoc routing protocols do not differentiate between these types of nodes and therefore cannot achieve optimal performance in hybrid WMNs. In this paper, we propose simple extensions to the Ad-hoc On-demand Distance Vector (AODV) routing protocol, which aim to take advantage of the heterogeneity in hybrid WMNs by preferentially routing packets via paths consisting of high capacity MESH_ROUTERS. In addition, we implement a simple channel selection scheme that reduces interference and maximizes channel diversity in multi-radio WMNs. Our simulation results show that in hybrid WMNs, our extensions result in significant performance gains over the standard AODV protocol.

Keywords: Wireless Mesh Networks, Routing, Hybrid Mesh Networks, Channel Diversity

1 Introduction

Wireless Mesh Networks (WMNs) are self-organizing and self-configuring wireless networks, typically implemented with IEEE 802.11 hardware. In conventional wireless LANs, clients communicate with access points via a single-hop wireless link and access points are interconnected via a wired backbone infrastructure. WMNs do not rely on such a wired backhaul and implement connectivity via a wireless multi-hop network. Their robustness, self-organizing and self-configuring nature, and the low cost of wide area deployment make WMNs an attractive platform for a wide range of applications, such as public safety and emergency response communications, intelligent transportation systems, or community networks.

*The authors are also affiliated with the School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane, QLD 4072, Australia.

Copyright ©2007, Australian Computer Society, Inc. This paper appeared at the Thirtieth Australasian Computer Science Conference (ACSC2007), Ballarat, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 62. Gillian Dobbie, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

We can differentiate between two types of nodes in a WMN: MESH_ROUTERS and MESH_CLIENTs. MESH_ROUTERS are relatively powerful and static nodes, which have either access to mains power or are equipped with high capacity batteries. In addition, MESH_ROUTERS typically have multiple radio interfaces, which significantly increases the transmission capacity if the radios are operated on orthogonal channels. In contrast to MESH_ROUTERS, MESH_CLIENTs are relatively resource constrained mobile client devices, such as WiFi-enabled PDAs. These devices usually have only a single radio interface and their key constraint is limited battery power.

A WMN that is entirely comprised of MESH_ROUTERS is referred to as an infrastructure WMN, whereas a client WMN is a network made up of client devices only (Akyildiz & Wang 2005). A client WMN is essentially identical to a pure mobile ad-hoc network (MANET) (Pirzada, McDonald & Datta 2006), and we can therefore consider WMNs a superset of MANETs. A hybrid WMN, such as illustrated in Fig. 1, consists of both MESH_ROUTERS and MESH_CLIENTs, with both types of nodes performing routing and forwarding functionality. In this case, MESH_ROUTERS form the (wireless) backbone of a hybrid WMN, whereas MESH_CLIENTs can be seen as a dynamic extension.

Current routing protocols for WMNs can be roughly grouped into two categories. The first category consists of protocols that are based on traditional routing protocols for wired networks such as RIP (Routing Information Protocol) or OSPF (Open Shortest Path First). Since these protocols are not able to handle node mobility or highly dynamic networks in general, their application is restricted to rel-

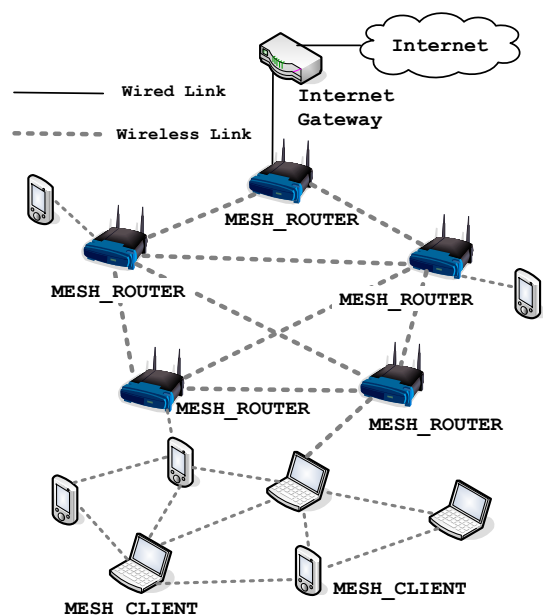


Figure 1: Hybrid Wireless Mesh Network

atively static infrastructure WMNs.

The second category consists of protocols that are based on MANET routing protocols. Tremendous research efforts have been made in this area over the last few years and an impressive number of mobile ad-hoc routing protocols has been proposed (Royer & Toh 1999). These protocols were designed for networks with highly mobile and typically power constrained devices. As a consequence, they are able to handle node mobility and the generally dynamic nature of client WMNs and hybrid WMNs, which is why they form the basis of most current WMN routing protocols.

However, since these routing protocols have been designed for relatively homogeneous MANETs, consisting entirely of resource constrained mobile devices, they do not perform optimally in highly heterogeneous hybrid WMNs. Current WMN routing protocols do not differentiate between different types of nodes (`MESH_CLIENTs` and `MESH_ROUTERS`) in the network and are therefore unable to take full advantage of high capacity `MESH_ROUTERS` in hybrid WMNs.

A fundamental problem of multi-hop wireless networks in general and WMNs in particular is the limited scalability and the degradation of performance with increasing path length. One approach to overcome this problem is to use multiple radio interfaces per node, operating on orthogonal channels. Multi-radio nodes have significantly increased capacity, due to reduced interference and the ability of full-duplex communication, which is not supported by single radio nodes. In order to achieve optimal performance in a multi-radio WMN, an efficient channel allocation and selection scheme is required. Even with complete knowledge of the network topology, optimal channel assignment is very difficult to achieve and is considered an NP-hard problem (Raniwala & Chiueh 2005).

The routing protocol presented in this paper aims to achieve two goals. First, it tries to make optimal use of high capacity `MESH_ROUTERS` in a hybrid WMN by routing packets along paths consisting of `MESH_ROUTERS` whenever possible. This not only increases the overall throughput and reduces latency, it also helps to conserve the battery power of client devices. Secondly, we present a very simple yet effective scheme that tries to maximize per-path channel diversity. For example, the scheme aims to prevent the use of the same channel on neighbouring links of an end-to-end path, which would lead to significant interference and performance degradation.

The Ad-hoc On-demand Distance Vector (AODV) routing protocol (Perkins, Royer & Das 2003) forms the basis of our work. AODV is a popular reactive MANET routing protocol with excellent scalability properties. Even though AODV has inherent support for multi-radio nodes, it lacks built-in support for optimal channel or interface selection and is therefore unable to maximize channel diversity. AODV has been designed for homogeneous MANETs where nodes have similar computational communication resources and are also similar in terms of their level and pattern of mobility. Thus, AODV will not be able to perform optimally if deployed on a hybrid WMN with high capacity `MESH_ROUTERS` that are static and are equipped with multiple radios.

In this paper, we present an extended version of AODV, called AODV-HM (AODV-Hybrid Mesh). The key contributions of our work are:

- We propose a simple modification of AODV's route discovery mechanism to allow selection of paths which maximize the use of `MESH_ROUTERS` and minimize the involvement of `MESH_CLIENTs`. These routes are more stable due to the lower mobility of `MESH_ROUTERS` and provide lower la-

tency, improved packet delivery rates and a lower control packet overhead.

- We integrate a channel selection scheme into AODV's route discovery mechanism which increases channel diversity of end-to-end paths. This reduces interference and contention and further increases the packet delivery rate and reduces latency.

The remainder of the paper is organized as follows: Section 2 discusses relevant related work. The AODV-HM protocol is explained in Section 3. In Section 4 we provide details of our simulation environment. Simulation results and their analysis are presented in Section 5 with concluding remarks in Section 6.

2 Related Work

2.1 Hyacinth

Hyacinth (Raniwala & Chiueh 2005) is a multi-channel static wireless mesh network protocol that uses multiple radios and channels to improve the network performance. It supports a fully distributed channel assignment algorithm, which can dynamically adapt to varying traffic loads. It uses a spanning-tree based routing algorithm to load balance the network as well as to rectify route failures. The `MESH_ROUTERS` having access to the wired network are considered as the root nodes of the spanning tree. Hyacinth's channel assignment algorithm breaks a single-channel collision domain into multiple collision domains, each operating on a different frequency. The channel assignment algorithm operates in two phases: Neighbour-Interface Binding and Interface-Channel Assignment. In the first phase each node separates its interfaces into UP-NICs and DOWN-NICs. Each node has control to change the channel on its DOWN-NICs only. During the second phase each node exchanges a periodic message, which contains the channel usage status, with its neighbours in the interference range. Using the per-channel total load information a node can issue a change channel message to its neighbour in order to switch to a least used channel. The advantage of this channel assignment scheme is that a fat-tree architecture is obtained in which links close to the root of the spanning tree are given higher bandwidth. The channel assignment is further integrated with the routing process. Each node having routing information to the root advertises this information to one-hop neighbours. This advertisement also contains the cost metric, which comprises of the hop-count and the residual uplink capacity. Each node receiving this advertisement makes a decision, based upon the cost, as regards to joining the advertising node. If the node decides to join, it sends an acceptance message to the advertising node and a departing message to the parent node with which it was previously attached. New nodes joining the network broadcast HELLO packets, such as to initiate the joining process by the neighbouring nodes.

2.2 Single-Radio Multi-Channel Routing Protocol

The Multi-Channel Routing Protocol (MCRP) (So & Vaidya 2004) is a routing protocol specifically designed for networks with single-radio nodes, which can support a channel switching delay of 80 μ s or less. The protocol assigns channels to data flows rather than assigning channels to nodes. This implies that all nodes supporting a flow have to be on one common channel. The advantage of this mechanism is

that once the route is established, nodes are not required to switch channels for the duration of the flow. The protocol considers all nodes in the network to be in essentially one of four states: free, locked, switching or hard-locked. The free nodes are nodes that are not supporting any flow at the moment. Locked nodes are those that are currently supporting one flow. A switching node is one that is supporting two or more flows on different channels. A hard-locked node is one, which due to certain constraints, cannot become a switching node. MCRP benefits from multiple channels without modifying the MAC protocol. The routing scheme is similar to that of AODV. However, each Route Request packet (RREQ) is broadcast on each channel in a round robin manner and each receiving node also rebroadcasts the RREQ. Intermediate nodes also create a Reverse Route to the source and maintain a channel number for the next hop with each Reverse Route table entry. To convey the channel information of the next hop, each RREQ contains the operating channel¹ number of the hop sending the RREQ. The RREQ also contains the channel table and flow table to be propagated along with each RREQ. The channel table contains the count of similar channels being consecutively used on a single flow path. The flow tables maintain a count of simultaneous flows being carried out on a single channel. These tables are used by the destination node to make a decision regarding the selection of the optimal route from multiple received RREQs. The Route Reply packet (RREP) is unicast from the destination to the source on the optimal path. All nodes forwarding the RREQ change their operating channels to the channel selected by the destination.

2.3 Multi-Radio Link Quality Source Routing

The Multi-Radio Link Quality Source Routing (MR-LQSR) (Draves, Padhye & Zill 2004) protocol has been developed by Microsoft for static community wireless networks. The protocol works in conjunction with the Mesh Connectivity Layer (MCL). The MCL permits higher layer applications to connect to the wireless mesh network using Wi-Fi or WiMAX. The MCL implements an interposition layer between the link and network layers. It essentially consists of a loadable driver, which acts as a virtual network adapter with the ability to multiplex several physical adapters. Routing of packets is carried out by MR-LQSR, which is an optimized version of the Dynamic Source Routing (DSR) protocol. The MR-LQSR protocol assumes that the number of wireless interfaces is equal to the number of channels being used in the network. The protocol identifies all nodes in the wireless mesh network and assigns weights to all possible links. To do so, the link information including channel assignment, bandwidth and loss rates are propagated to all nodes in the network. This propagation is combined with the delivery of DSR control packets. The Expected Transmission Time (ETT) on each link is computed using the Expected Transmission Count (ETX), bandwidth and packet loss. The ETT metric is further used to compute the Weighted Cumulative Expected Transmission Time (WCETT), which defines the path metric designed for multi-radio WMNs. The WCETT is then applied to the Link Cache scheme of the DSR protocol. In native DSR, the default cost of links is set to one. Hence, when the Dijkstra algorithm is executed over the link cache by a source node, the shortest path in terms of number of hops is always returned. However, when the

WCETT is used as the link cost, the protocol aims to return the path in terms of link bandwidth, loss rate and channel diversity.

2.4 Multi-Channel Routing Protocol

The Multi-Channel Routing (MCR) protocol (Kyasanur & Vaidya 2006) has been developed for dynamic WMNs, where nodes have multiple wireless interfaces, each supporting multiple channels. The protocol makes use of an interface switching mechanism to assign interfaces to channels. Two types of interfaces are assumed: fixed and switchable. In fixed interfaces, K number of interfaces out of a total M interfaces are assumed to be operating on K fixed channels. In the switchable interfaces, the remaining interfaces are dynamically assigned to any of the remaining channels. Switching is carried out depending upon the maximum number of data packets queued for a single channel. Multiple queues are maintained for all switchable interfaces. Each node maintains a neighbour table and a channel usage list. The neighbour table contains information regarding the fixed channels used by the node's neighbours. The channel usage list contains the count of nodes that are using each channel as their fixed channel. Each node periodically transmits a HELLO packet on all channels, containing the node's fixed channel number. Each node receiving the HELLO packet updates its neighbour table and channel usage list. The information from the table and list is used to control the channel and interface switching mechanism. The switching mechanism assists the MCR protocol in finding routes over multiple channels. MCR uses a new routing metric, which is computed as a function of channel diversity, interface switching cost and hop-counts. The diversity cost is assigned according to the least number of channels used in a route. Thus, a route with a larger number of distinct channels in a route is considered to be having a lower diversity cost. The switching cost is used to minimize the frequent switching of wireless interfaces. The route discovery mechanism of MCR is similar to that of DSR. In addition, each RREQ also contains the channel number and switching cost. When the destination receives the RREQ, it computes the diversity cost (number of channels in the RREQ) and the switching cost (sum of all link switching costs). These costs help the destination in determining and selecting the optimal path available between the source and the destination.

Table 1 presents a comparison of the discussed protocols. All protocols assume homogeneous node configurations i.e. equal number of interfaces and do not differentiate between different node types. Hyacinth and MR-LQSR have been specifically designed for infrastructure WMNs and have no support for client mobility. Hyacinth, MR-LQSR and MCR use interface switching to improve upon the routing performance in the network, where the interfaces are switched dynamically to different channels. MCRP makes use of channel switching on a single interface to connect to nodes operating on the same channel, to improve upon the network performance. Both interface and channel switching achieve efficient use of the available spectrum. However, the virtual switching protocol and the constant switching incurs significant delays causing excessive jitter (Draves et al. 2004). In the remainder of this paper, we will present and discuss AODV-HM, which can achieve efficient spectrum usage without the need for a complex and expensive channel switching mechanism.

¹The operating channel of a node is the default channel on which it is generally listening on.

Table 1: Comparison of Recent WMN Protocols

Protocols	Research Group	Year	Type	Mobility	Derivative	Routing Metric	Remarks
Hyacinth	Stony Brook University	2005	Multi-Radio	No	Spanning Tree	Hop Count, link/path loads	Independent channel switching protocol required.
MCRP	University of Illinois, Urbana-Champaign	2004	Single-Radio	Yes	AODV	Hop Count	High speed interface switching required. Complex to handle multiple flows.
MR-LQSR	Microsoft Research	2004	Multi-Radio	No	DSR	WCETT	Proprietary Mesh Connectivity Layer. Bandwidth computed using packet probes. Requires propagation of intermediary node ETT's to source node.
MCR	University of Illinois, Urbana-Champaign	2006	Multi-Radio	Yes	DSR	Channel diversity and switching cost	Hybrid of fixed and switchable channels. Periodic distribution of channel usage lists.

3 Mesh-Aware AODV Protocol

3.1 Standard AODV

The AODV is inherently a distance vector routing protocol that has been optimised for ad-hoc wireless networks. It is an on demand or reactive protocol, as it finds the routes only when required. AODV borrows basic route establishment and maintenance mechanisms from the DSR protocol, and hop-to-hop routing vectors from the Destination-Sequenced Distance-Vector (DSDV) routing protocol. To avoid the problem of routing loops, AODV makes extensive use of sequence numbers in control packets. When a source node intends to communicate with a destination node whose route is not known, it broadcasts a Route Request packet (RREQ). Each RREQ contains an ID, source and destination node IP addresses, and sequence numbers together with a hop count and control flags. The ID field uniquely identifies the RREQ; the sequence numbers indicate the freshness of control packets and the hop-count maintains the number of nodes between the source and the destination. Each recipient of the RREQ that has not seen the source IP and RREQ ID pair or does not have a fresher (with larger sequence number) route to the destination rebroadcasts the same packet after incrementing the hop-count.

Intermediate nodes create and preserve a Reverse Route to the source node for a certain interval of time. When the RREQ reaches the destination node or any node that has a fresh route to the destination, a Route Reply packet (RREP) is generated and unicast back to the source of the RREQ. Each RREP contains the destination sequence number, the source and the destination IP addresses, route lifetime, and a hop count and control flags.

Each intermediary node that receives the RREP increments the hop-count, establishes a Forward Route to the source of the packet, and transmits the packet via the Reverse Route. To preserve connectivity information, each node executing AODV can use link-layer feedback or periodic HELLO packets to detect link breakages to nodes that it considers as its immediate neighbours. In case a link break is detected for a next hop of an active route, a Route Error packet (RERR) is sent to its active neighbours that were using that particular route.

When using AODV in a network with multi-radio nodes, each RREQ is broadcast on all interfaces. In order to avoid broadcast storms, a random delay is added in the transmission of each RREQ. Intermedi-

ate nodes with one or more interfaces operating on a shared channel, receive the RREQ and create a Reverse Route that points towards the source node. If the RREQ is a duplicate, it is simply dropped. The first received RREQ received by the destination or any intermediary node is selected and all other RREQs are discarded. The RREP is generated in response to the selected RREQ, and is sent back to the source node on the existing Reverse Route.

3.2 AODV-HM

We have made the following assumptions for the design and evaluation of our AODV-HM protocol:

- All MESH_CLIENTs and MESH_ROUTERs should have at least one common operating channel.
- The transmission and reception ranges of the wireless transceivers are comparable.
- The wireless antennas are omni-directional.

The aim of AODV-HM is to maximize the involvement of MESH_ROUTERs into the routing process without significantly lengthening the paths. In addition, we want to maximize channel diversity in the selected paths. To implement these features we make two changes to the RREQ header. First, we add a 4-bit counter (MR-Count) indicating the number of MESH_ROUTERs encountered on the path taken by the RREQ. We further add a 7-bit field (Rec-Chan), which advertises the optimal channel to be used for the Reverse Route. We have used the existing 11 reserved bits in the RREQ header. The first four bits represent the MR-Count while the remaining seven represent the Rec-Chan as shown in Fig. 2.

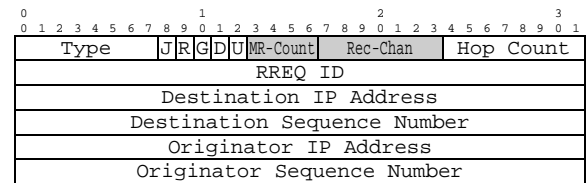


Figure 2: AODV-HM Route Request Packet Header

3.2.1 Discovery of MESH_ROUTERs

Whenever a MESH_CLIENT intends to communicate with another node whose route is not available in

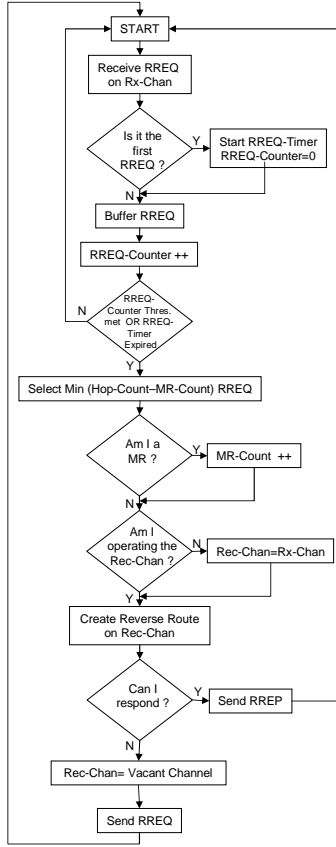


Figure 3: RREQ Processing in AODV-HM

its routing table, it broadcasts a RREQ on all of its interfaces. Prior to the broadcast, it also sets the MR-Count in the RREQ to zero. Each MESH_ROUTER forwarding the RREQ increments the MR-Count field by one. When the RREQ is received by the destination or any intermediary node that can respond, the process shown in Fig. 3 is initiated.

If the RREQ has not been received earlier², a RREQ-Timer is started and a RREQ-Counter is initialized. The RREQ-Timer determines the amount of time a node should wait after receipt of the first RREQ and before forwarding the optimal RREQ. The RREQ-Timer helps to evaluate alternate copies of the same RREQ arriving via different paths. The RREQ-Counter maintains a count of these copies. All copies of the RREQ are then buffered until the time when either the RREQ-Timer expires or the RREQ-Counter reaches a certain threshold.

The optimal values for the RREQ-Timer and Counter are primarily dependent upon the average node density. In case the density is high, the RREQ-Counter will reach its threshold value well within the RREQ-Timer. However, in case of a sparse network, the RREQ-Counter may never reach its threshold value before the RREQ-Timer expires. In the standard AODV protocol, the minimum Route Discovery Latency (RDL), i.e. time between transmission of the first RREQ and the receipt of its corresponding RREP, is

$$RDL = 2 \times n_p \times \text{NODE_TRAVERSAL_TIME}$$

where n_p is the number of nodes on the path (excluding the source node) taken by the RREP. NODE_TRAVERSAL_TIME is the approximate time taken by a packet to pass through one node.

For lower RREQ-Counter values in high density networks, AODV-HM incurs a RDL similar to that

of the standard AODV. However, in sparse networks with large RREQ-Counter values, AODV-HM may incur a maximum RDL of:

$$RDL = n_p \times \text{NODE_TRAVERSAL_TIME} + n_p \times \text{RREQ-Timer}$$

In order to minimize the RDL, a low value of the RREQ-Counter is maintained or the RREQ-Timer is kept as close to the NODE_TRAVERSAL_TIME as possible.

When the RREQ-Timer expires or the RREQ-Counter threshold is reached, the RREQ, for which the routing metric (Hop-Count - MR-Count) is minimal, is selected. This selection is done from the set of n RREQs, stored in the RREQ Buffer (RREQ-BUFF), as indicated in Equation 1.

$$\text{RREQ}_s = \text{RREQ}_i \mid \min_{1 \leq i \leq n} (\text{Hop-Count}_i - \text{MR-Count}_i) \quad (1)$$

Let's consider the scenario shown in Fig. 4, where MESH_CLIENT-5 (Source) wants to communicate with MESH_CLIENT-45 (Destination).

The darker nodes represent the MESH_ROUTERS and the remaining nodes are MESH_CLIENTS. Standard AODV does not distinguish between MESH_ROUTERS and MESH_CLIENTS. Accordingly, when a route discovery is initiated from MESH_CLIENT-5 for MESH_CLIENT-45, the first arriving RREQ at MESH_CLIENT-45 establishes the route. The first route, between the source and destination, established using standard AODV is represented as follows:

$$R_{SD1} = (5 \rightarrow 63 \rightarrow 59 \rightarrow 55 \rightarrow 37 \rightarrow 45)$$

Route R_{SD1} has a hop-count of five and contains three intermediary MESH_ROUTERS and one MESH_CLIENT. However, there may be occasions where the first RREQ arriving at the destination contains no MESH_ROUTERS, e.g. $R_{SD} = 5 \rightarrow 22 \rightarrow 14 \rightarrow 35 \rightarrow 37 \rightarrow 45$. If MESH_CLIENTs operate on a single channel, as is typically the case, the above scenario would lead to a significant performance degradation over a route consisting only of MESH_CLIENTs (Li, Blake, Couto, Lee & Morris 2001).

In contrast, AODV-HM is able to create Reverse Routes that traverse predominantly MESH_ROUTERS by delaying RREQs at intermediary nodes and selectively forwarding the one consisting mostly of MESH_ROUTERS, instead of the first one to arrive. For example, MESH_ROUTER-60 is likely to receive more than one RREQs originating from MESH_CLIENT-5. In case the first RREQ reaches MESH_ROUTER-60 via

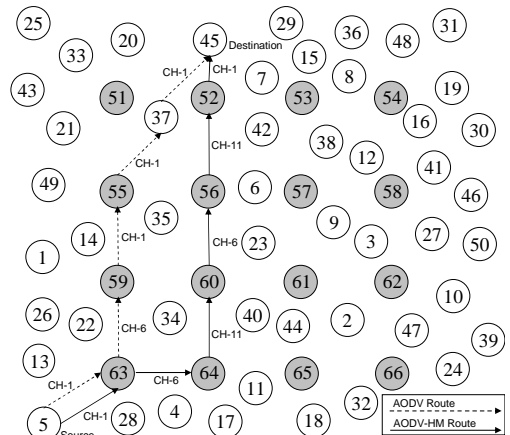


Figure 4: Route Development in AODV-HM

²Determinable through the Source IP and RREQ ID mapping.

MESH_CLIENT-34 (with MR-Count=1), and the RREQ reaches MESH_ROUTER-60 via MESH_ROUTER-64 (with MR-Count=2), AODV-HM would select the latter which contains the smaller number of MESH_CLIENTs. In standard AODV, MESH_ROUTER-60 would simply forward the first RREQ received from MESH_CLIENT-34.

Similarly, more than one RREQs are received by the destination MESH_CLIENT-45. Let's assume five RREQs from MESH_CLIENT-5 have reached MESH_CLIENT-45 and stored in the RREQ-BUFF. The paths taken by the five RREQs are as follows:

$$\begin{aligned} R_{SD1} &= 5 \rightarrow 63 \rightarrow 34 \rightarrow 35 \rightarrow 37 \rightarrow 45 \\ R_{SD2} &= 5 \rightarrow 63 \rightarrow 59 \rightarrow 55 \rightarrow 51 \rightarrow 45 \\ R_{SD3} &= 5 \rightarrow 63 \rightarrow 59 \rightarrow 55 \rightarrow 51 \rightarrow 20 \rightarrow 45 \\ R_{SD4} &= 5 \rightarrow 63 \rightarrow 64 \rightarrow 60 \rightarrow 56 \rightarrow 52 \rightarrow 45 \\ R_{SD5} &= 5 \rightarrow 63 \rightarrow 64 \rightarrow 40 \rightarrow 6 \rightarrow 7 \rightarrow 45 \end{aligned}$$

Now using Eq. 1, we get the minimum difference between the Hop-Count and MR-Count for the above routes as follows: $R_{SD1} : 5 - 1 = 4$, $R_{SD2} : 5 - 4 = 1$, $R_{SD3} : 6 - 4 = 2$, $R_{SD4} : 6 - 5 = 1$ and $R_{SD5} : 6 - 2 = 4$ respectively. The minimum cost is achieved using the RREQ that arrived via routes R_{SD2} and R_{SD4} . In case two or more RREQs have the same cost, the first of these to arrive is responded to and the corresponding route is established.

3.2.2 Channel Diversity

In a wireless network, as the physical medium is shared, nodes have to constantly contend with each other to gain access to the network. All nodes before making a transmission execute the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) protocol to avoid future collisions (IEEE 1997). The effectiveness of the CSMA/CA protocol is influenced by the density, mobility and traffic pattern of the network (Bianchi 2000). In order to minimize packet collisions and contention, the physical medium is generally segregated using non-interfering channels. This in turn reduces the number of nodes contending per channel, which also lowers the number of packet collisions.

As mentioned earlier, the standard AODV protocol typically adds some random delay prior to the transmission of RREQs over multiple interfaces. Thus, the Reverse and Forward Routes may or may not have similar channel assignments³. For example, in Fig. 4 the route between MESH_CLIENT-5 and MESH_CLIENT-45 indicated with dashed arrows includes three MESH_ROUTERS: 55, 59 and 63. Let's assume that the MESH_CLIENTs have one radio each operating on Channel 1 (CH-1) and that the MESH_ROUTERS have three radios each, operating on Channel 1 (CH-1), Channel 6 (CH-6) and Channel 11 (CH-11) respectively. During the route discovery process, MESH_ROUTER-63 introduces a small random delay before forwarding the RREQ on each of its three channels. If we assume the smallest delay is selected for CH-6, MESH_ROUTER-59 receives the first RREQ via this channel. In this case, the Reverse Route is created to MESH_CLIENT-5 via CH-6. When MESH_ROUTER-59 retransmits the RREQ, it employs a similar mechanism. However, this time the first RREQ to reach MESH_ROUTER-55 is via CH-1. Thus, MESH_ROUTER-55 creates the Reverse Route to the source MESH_CLIENT-5 via CH-1. Similarly, the Reverse Route from MESH_CLIENT-37 is created via CH-1. This essentially introduces another collision domain between MESH_ROUTER-59 and MESH_CLIENT-37.

³The first RREQ received on any interface determines the channel used for the Reverse Route to the source node.

Table 2: Assignment of Recommended Channels

IEEE Standard (3-bits)	Channel Number (4-bits)
802.11a	000
802.11b	001
802.11g	010
...	...
	1 ~ 16

Packets sent from MESH_ROUTER-59 to MESH_ROUTER-55, as well as packets sent from MESH_ROUTER-55 to MESH_CLIENT-37 have to contend for the same medium since they all share the same common channel (CH-1). Thus, the worst case channel selection strategy scenario for a route traversing through nodes with multiple radios may degrade to that of a path comprised of single-radio nodes.

In AODV-HM, we use a simple mechanism to achieve effective channel assignment during route discovery. Each node, before propagating a RREQ, appends the Recommended Channel (Rec-Chan) to the RREQ, as shown in Fig. 2 and Fig. 3. The Rec-Chan informs the RREQ recipient about the desired channel to be used for creating the Reverse Route. The Rec-Chan value is implemented as a 7 bit number, and its value is set according to Table 2. The first three bits define the IEEE physical layer standard the radio is operating on. The following 4 bits indicate the specific channel number. For example, in the network shown in Fig. 4, all MESH_CLIENTs are operating a single radio and are tuned to CH-1 of 802.11b. In this case, Rec-Chan will have a value of 17 (0010001). If a node operates only one radio, the Vacant Channel is the current operating channel. A node with multiple radios has the discretion to recommend any Vacant Channel. The Vacant Channel is selected based upon the following two criteria:

- The Rec-Chan is not interfering with the channel being used on the Reverse Route.
- The Rec-Chan is the least loaded channel.

A RREQ can be received by a multi-radio node on one of its Receive Channels (Rx-Chan). However, depending upon the current assignment of channels to the interfaces, the Rx-Chan may or may not be equal to the Rec-Chan. For example, a node could have all of its radios tuned to channels other than the Rec-Chan, which would make it impossible create a Reverse Route using the Rec-Chan. In case a node has an interface operating on Rec-Chan, it creates the Reverse Route to the previous hop using that interface, otherwise it creates the Reverse Route via the Rx-Chan.

Coming back to the example of Fig. 4, before initiating the RREQ, MESH_CLIENT-5 sets the Rec-Chan to its current operating channel, i.e. CH-1. As MESH_CLIENT-5 is a single radio node, the RREQ is received by MESH_ROUTER-63 on CH-1 only. In this case Rx-Chan is equal to Rec-Chan, so MESH_ROUTER-63 creates the Reverse Route to MESH_CLIENT-5 using CH-1. MESH_ROUTER-63 is operating on three channels, i.e. CH-1, CH-6 and CH-11. Using the criteria mentioned above, it now selects the Vacant Channel to be equal to CH-6. The Rec-Chan is then set to the Vacant Channel and the RREQ is broadcast over all three interfaces. MESH_ROUTER-64, which is also operating the same three channels, now receives three RREQs from MESH_ROUTER-63. Since the Rec-Chan in all three RREQs is CH-6, MESH_ROUTER-64 creates the Reverse Route to MESH_ROUTER-63 using CH-6.

In our example, MESH_CLIENTs are operating on CH-1 and so this channel would experience a relatively high load. Thus, MESH_ROUTER-64 selects and recommends the Vacant Channel to be CH-11, which

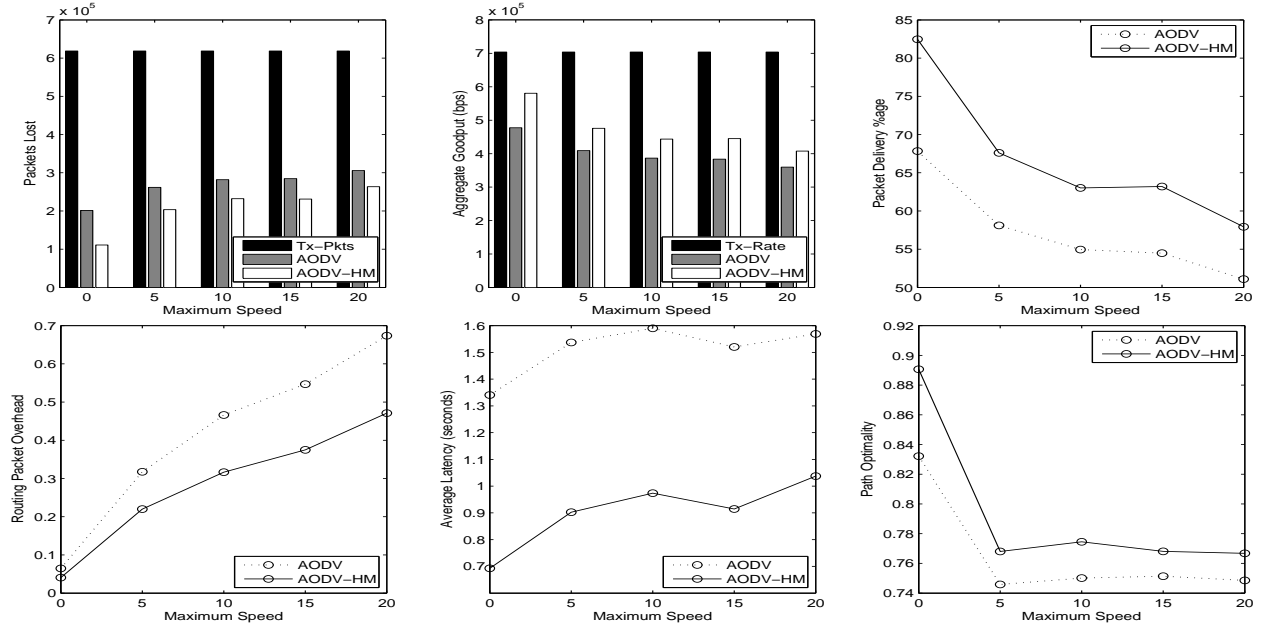


Figure 5: Results of Simulation 1

also does not interfere with the channel used on the Reverse Route. Similarly, MESH_ROUTER-60 creates the Reverse Route via CH-11 and recommends the Vacant Channel CH-6. In this manner, AODV-HM creates a route between MESH_CLIENT-5 and MESH_CLIENT-45 with less interference and contention compared to the route that is selected by standard AODV.

The routing metric used in AODV-HM aims to minimize the number of MESH_CLIENTs present in a particular route. This may seem analogous to shortest path routing, but this is not the case, since the metric attempts to route traffic through the MESH_ROUTERS, which in turn maximize the stability and channel diversity of the routes. A number of other routing metrics like ETX, ETT and WCETT also exist. Of these metrics, only WCETT takes advantage of the channel diversity, however, its direct application to AODV, which is a distance vector routing protocol, requires extensive modifications to the protocol's inherent working (Ramachandran, Buddhikot, Chandranmenon, Miller, Belding-Royer & Almeroth 2005).

4 Simulation Environment

We evaluated the efficiency of the AODV-HM protocol through extensive simulations in NS-2 (NS 1989), using the Extended Network Simulator (ENS) extensions (Raman & Chebrolu 2005). A WMN covering an area of 1 square km is established using uniformly distributed static MESH_ROUTERS and randomly distributed mobile MESH_CLIENTs. Concurrent UDP connections are established between randomly selected source and destination MESH_CLIENT pairs. A total of four simulations were conducted to evaluate the performance of the AODV-HM protocol under varying mobility, traffic load and node configurations. The parameters common to all four simulations are listed in Table 3. The simulations provide the following performance metrics:

Packets Lost: The number of data packets that were lost due to unavailable or incorrect routes, MAC layer collisions or through the saturation of interface queues.

Table 3: Simulation Parameters

Examined protocols	AODV and AODV-HM
Simulation time	900 seconds
Simulation area	1000 x 1000 m
Propagation model	Two-ray Ground Reflection
Mobility model for MESH_CLIENTs	Random waypoint
Maximum speed of MESH_CLIENTs ⁴	1 m/s
Transmission range	250 m
Number of connections ⁴	30
Traffic type	CBR (UDP)
Packet size	128 bytes
Packet rate	25 pkts/sec
Number of MESH_ROUTERS ⁴	25
Number of MESH_ROUTER Interfaces ⁴	3
Number of MESH_CLIENTs	50
Number of MESH_CLIENT Interfaces	1
MESH_CLIENT RREQ-Counter	5 packets
MESH_ROUTER RREQ-Counter	25 packets
MESH_CLIENT RREQ-Timer	50 ms
MESH_ROUTER RREQ-Timer	250 ms

Aggregate Goodput: The number of data bits successfully transmitted in the network per second.

Packet Delivery Percentage: The ratio between the number of data packets successfully received by destination nodes and the total number of data packets sent by source nodes.

Routing Overhead: The ratio of the total number of control packets generated to the total number of received data packets.

Average Latency: The mean time in seconds taken by data packets to reach their respective destinations.

Path Optimality: The ratio between the length (number of hops) of the shortest possible path and the actual path taken by data packets.

5 Results and Analysis

5.1 Simulation 1 : Varying the MESH_CLIENT Speeds

In Simulation 1, we have varied the maximum speed of the MESH_CLIENTs from 0 m/s to 20m/s, with increments of 5 m/s. The results, shown in Fig. 5, indicate that the packet loss is consistently lower for

⁴The values of these parameters are varied in Simulations 1, 2, 3 and 4 respectively.

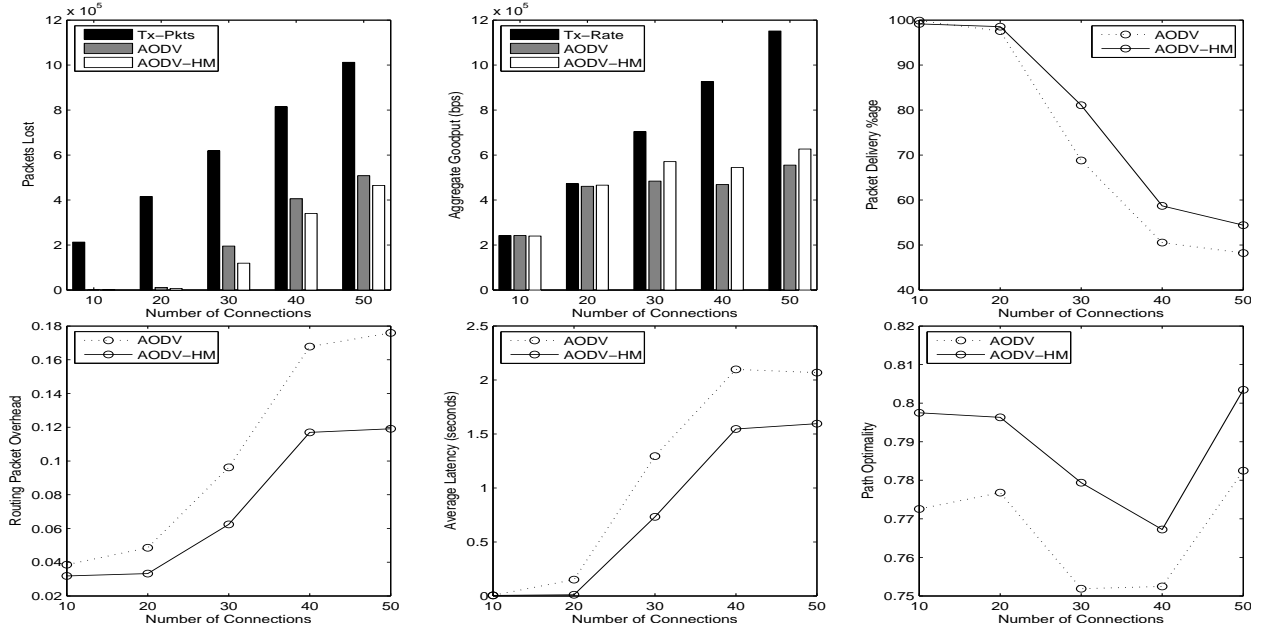


Figure 6: Results of Simulation 2

AODV-HM compared to standard AODV. This is primarily due to the selection of static `MESH_ROUTERS` in the routing process, which offer more stable routes with less contention. On the other hand, the standard AODV has no option for prioritizing the routing according to the node type. Thus both the `MESH_ROUTERS` and `MESH_CLIENTS` are randomly selected in establishing a route. Routes consisting mostly of single-radio `MESH_CLIENTS` have a higher packet loss due to the extended contention for the wireless medium, which can lead to saturated interface queues and packets being dropped. The routes formed by AODV-HM may also involve `MESH_CLIENTS` in its paths, but their number is relatively smaller. The lower number of `MESH_CLIENTS` in the path means improved utilization of the channel diversity and lower contention for the wireless medium. This in effect reduces the packet drop when the AODV-HM protocol is engaged. However, when the `MESH_CLIENTS` move at a higher speed, the routes are frequently broken and recreated. Thus we see an increase in the number of packets lost with the increase in the network mobility.

The number of packets lost in the network, due to collisions or saturation of interface queues, directly influences the aggregate goodput of the network. AODV-HM shows improved goodput over standard AODV for all speeds. Even though AODV-HM aims to route traffic through the `MESH_ROUTERS`, at higher speeds the routes become extremely unstable due to the movement of the source, destination and intermediary `MESH_CLIENTS`. The packet delivery rate of AODV-HM ranges from 83% at zero mobility to almost 57% at a speed of 20 m/s. Nevertheless, the packet delivery of AODV-HM is consistently higher than for standard AODV.

AODV-HM has the ability to create more stable routes by preferably involving static `MESH_ROUTERS`. This in turn reduces the number of route discoveries in the network, thereby lowering the control packet overhead. In addition, as AODV-HM is able to achieve a higher packet delivery rate, the control packet overhead per received data packet is significantly lower than for AODV. However, it should be noted that AODV-HM does not incur any additional byte overhead, since the MR-Count and Rec-Chan fields occupy existing fields of the AODV RREQ

header. The average latency of the network using AODV-HM is considerably lower than that of the standard AODV at varying speeds. The lower latency highlights the success of AODV-HM's route selection mechanism along with the dynamic channel assignment carried out during the route discoveries. As mentioned earlier, standard AODV selects the first incoming RREQ. However, the first RREQ to arrive does not necessarily arrive via the shortest path (in terms of the number of hops). Our simulations show that by delaying the RREQs in AODV-HM, the chance of discovering shorter paths is increased. This is shown in the path optimality metric, which shows that AODV-HM paths have higher path optimality, i.e. they are shorter.

5.2 Simulation 2 : Varying the Traffic Load

In Simulation 2, we varied the traffic load in the network by increasing the number of simultaneous connections between the `MESH_CLIENTS` from 10 to 50, with an increment of 10 connections. Our results (Fig. 6) show that at lower traffic loads, the performance of AODV-HM is comparable to that of the standard AODV protocol. However, as the load is increased, the packet loss incurred by AODV increases significantly, thereby decreasing the goodput of the network. The packet delivery rate for both protocols stays close to 100% up to 20 concurrent connections. Beyond this point, the packet delivery rate of both protocols starts to degrade. Since the routes created by AODV-HM contain more `MESH_ROUTERS` than those created using AODV, we see an improved performance of the former under increasing traffic loads, relatively to AODV. The routing packet overhead of AODV-HM also remains lower. The latency of the network increases with the increase in the traffic load due to increasing contention for the wireless medium by nodes operating on interfering channels. However, AODV-HM still manages to maintain a significant improvement over AODV.

5.3 Simulation 3 : Varying the Number of MESH_ROUTERS

We have simulated hybrid WMNs with varying numbers of `MESH_ROUTERS`: 0, 4, 9, 16 and 25. Interest-

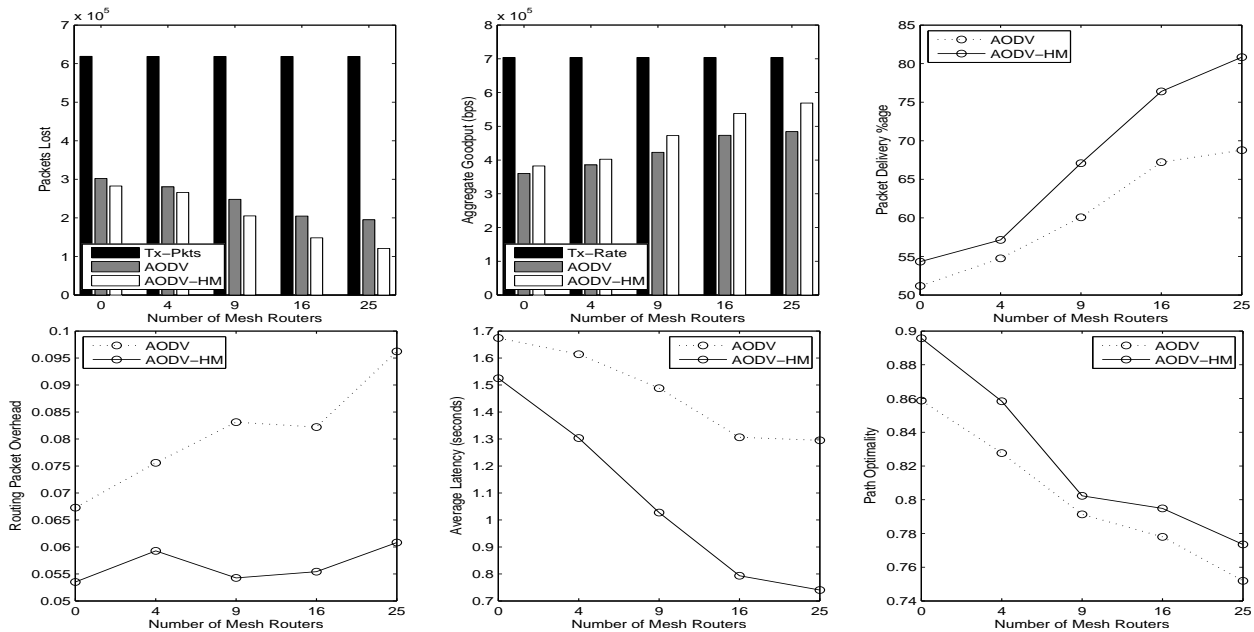


Figure 7: Results of Simulation 3

ingly, the results (Fig. 7) show that even when no `MESH_ROUTER` is present in the network, AODV-HM still has a lower packet loss than standard AODV. This is because AODV-HM delays the received `RREQ` and responds to the one with the lowest hop count, since in this case `MR-Count=0`. In contrast, standard AODV responds to the first `RREQ` that it received, which may not necessarily have arrived via the shortest path, due to interference or contention on one of the links. This use of non-shortest paths in AODV increases the total load in the network and increases contention and packet loss. This is also confirmed by looking at the path optimality of AODV-HM, which is much closer to the shortest possible path⁵ than the paths created by AODV. The performance of both protocols improves with an increasing number of `MESH_ROUTERS` in the network. However, AODV-HM makes more efficient use of the `MESH_ROUTERS` and achieves an improved packet delivery rate, decreased packet overhead, and significantly lower latency.

5.4 Simulation 4 : Varying the Number of Radios on each `MESH_ROUTER`

In Simulation 4, we varied the number of radio interfaces in each `MESH_ROUTER` from 1 to 9, with increments of 2 interfaces. All channels have been configured to be orthogonal and non-interfering with each other. The results of Simulation 4 (Fig. 8) reveal that the only time standard AODV outperforms AODV-HM in terms of packet delivery rate is when all nodes are limited to a single radio operating on the same channel. This means that forcing packets to go via `MESH_ROUTERS`, when they do not have a higher capacity than `MESH_CLIENTS`, can have a slightly negative impact. In this particular case, `MESH_ROUTERS` cannot take advantage of the channel diversity mechanisms of AODV-HM if they are equipped with only a single interface. Nevertheless, since the `MESH_ROUTERS` are static, they provide more stable routes than mobile `MESH_CLIENTS`, which results in a reduced packet overhead and lower latency in AODV-HM. The packet delivery rate rapidly improves when the number of

interfaces in the `MESH_ROUTERS` is increased. AODV-HM significantly outperforms standard AODV in these scenarios. However, increasing the number of `MESH_ROUTER` interfaces to more than three does not show any further improvements. This is due to the fact that three interfaces operating on orthogonal channels are sufficient to provide the required capacity and channel diversity for the network and traffic pattern considered in our simulation. However, the ideal number of `MESH_ROUTER` interfaces will vary for different types of networks with different size, density and traffic load. AODV-HM maintains its superior performance over the standard AODV protocol with the increase in the number of interfaces. It shows significantly lower packet overhead and latency, and a considerably better packet delivery ratio.

6 Conclusions

Hybrid WMNs consist of a mix of mobile `MESH_CLIENTS` and static `MESH_ROUTERS`. These two types of node differ considerably in terms of their capacity to forward packets. `MESH_ROUTERS` are typically much less resource constrained than mobile `MESH_CLIENTS`, and can be assumed to be equipped with multiple radio interfaces. Current WMN routing protocols do not differentiate between the types of node in a WMN, and are therefore not able to exploit the inherent heterogeneity in hybrid WMNs. In this paper, we presented simple extensions to the AODV routing protocol to increase its efficiency in hybrid WMNs. We defined a new routing metric that allows more efficient use of high capacity `MESH_ROUTERS` by preferential routing of packets via paths traversing the `MESH_ROUTERS`. In addition, we integrated a channel or interface selection scheme to maximize channel diversity and therefore minimize interference on end-to-end paths. We have performed extensive simulations to evaluate the performance of AODV-HM and compared it with standard AODV. The results show that AODV-HM consistently outperforms AODV in terms of all our performance metrics and for all simulation scenarios, except for the one special case discussed above. Compared to AODV, AODV-HM achieves an increase in the packet delivery rate of up to 15% in absolute terms, and achieves a reduction in latency by up to 50%. These are encouraging results, given that

⁵The shortest possible path is determined by an omniscient entity present in the NS-2 simulator known as the General Operations Director.

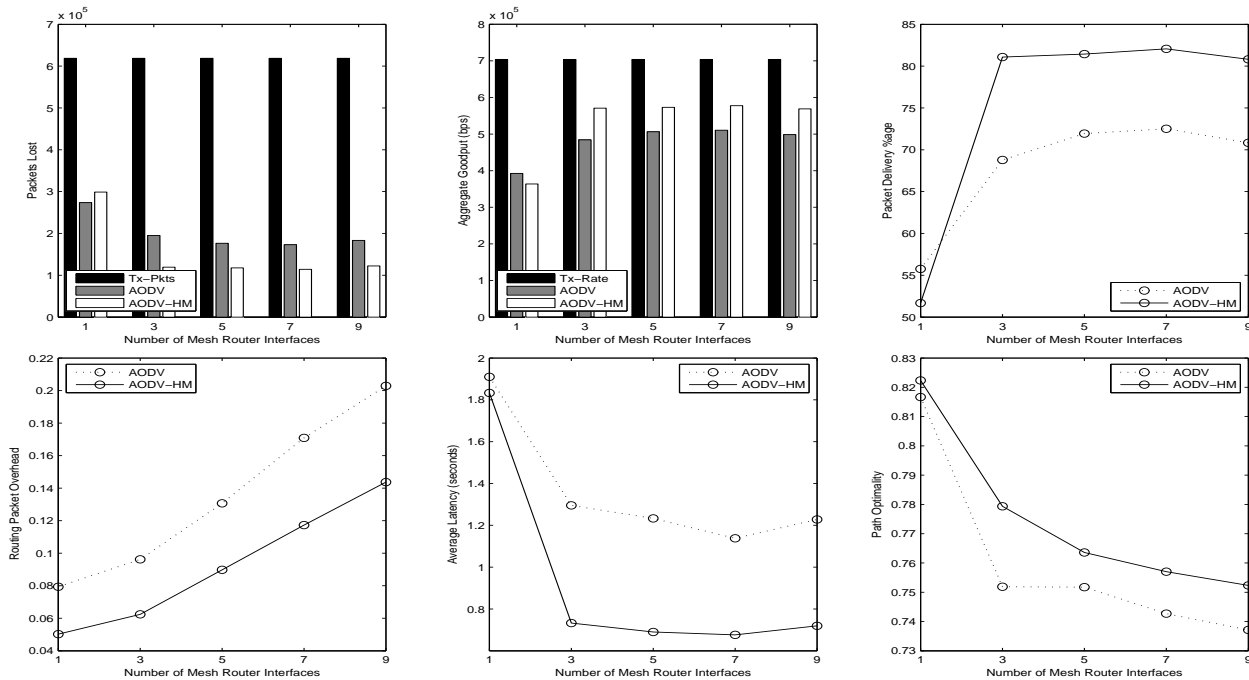


Figure 8: Results of Simulation 4

our proposed changes to the AODV protocol are very simple and incur only very minor additional overhead or complexity.

Acknowledgements

National ICT Australia is funded by the Australian Government's Department of Communications, Information Technology, and the Arts and the Australian Research Council through Backing Australia's Ability and the ICT Research Centre of Excellence programs and the Queensland Government.

References

- Akyildiz, I. F. & Wang, X. (2005), 'A Survey on Wireless Mesh Networks', *IEEE Communications Magazine* **43**(9), S23–S30.
- Bianchi, G. (2000), 'Performance Analysis of the IEEE 802.11 Distributed Coordination Function', *IEEE Journal on Selected Areas in Communications* **18**(3), 535–547.
- Draves, R., Padhye, J. & Zill, B. (2004), Routing in Multi-Radio, Multi-Hop Wireless Mesh Networks, in 'Proceedings of the 10th Annual International Conference on Mobile Computing and Networking', ACM Press, pp. 114–128.
- IEEE (1997), 'Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications 802.11'.
- Kyasanur, P. & Vaidya, N. H. (2006), 'Routing and Link-layer Protocols for Multi-Channel Multi-Interface Ad Hoc Wireless Networks', *SIGMOBILE Mobile Computing and Communications Review* **10**(1), 31–43.
- Li, J., Blake, C., Couto, D. S. J. D., Lee, H. I. & Morris, R. (2001), Capacity of Ad Hoc Wireless Networks, in 'Proceedings of the 7th Annual International Conference on Mobile Computing and Networking', ACM Press, pp. 61–69.
- NS (1989), 'The Network Simulator', <http://www.isi.edu/nsnam/ns/>.
- Perkins, C., Royer, E. M. & Das, S. (2003), 'Ad hoc On-Demand Distance Vector (AODV) Routing', *IETF RFC 3561*.
- Pirzada, A. A., McDonald, C. & Datta, A. (2006), 'Performance Comparison of Trust-Based Reactive Routing Protocols', *IEEE Transactions on Mobile Computing* **5**(6), 695–710.
- Ramachandran, K., Buddhikot, M., Chandranmenon, G., Miller, S., Belding-Royer, E. & Almeroth, K. (2005), On the Design and Implementation of Infrastructure Mesh Networks, in 'Proceedings of the IEEE Workshop on Wireless Mesh Networks (WiMesh)', IEEE Press.
- Raman, B. & Chebrolu, C. (2005), Design and Evaluation of a new MAC Protocol for Long-Distance 802.11 Mesh Networks, in 'Proceedings of the 11th Annual International Conference on Mobile Computing and Networking (MobiCom)', ACM Press, pp. 156–169.
- Raniwala, A. & Chiueh, T. C. (2005), Architecture and Algorithms for an IEEE 802.11-based Multi-Channel Wireless Mesh Network, in 'Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)', Vol. 3, IEEE Press, pp. 2223–2234.
- Royer, E. M. & Toh, C. K. (1999), 'A Review of Current Routing Protocols for Ad hoc Mobile Wireless Networks', *IEEE Personal Communications Magazine* **6**(2), 46–55.
- So, J. & Vaidya, N. H. (2004), A Routing Protocol for Utilizing Multiple Channels in Multi-Hop Wireless Networks with a Single Transceiver, Technical report, Dept. of Computer Science and Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.

Searching with Style: Authorship Attribution in Classic Literature

Ying Zhao

Justin Zobel

School of Computer Science and Information Technology
RMIT University,
GPO Box 2476, Melbourne, Australia,
Email: yizhao, jz@cs.rmit.edu.au

Abstract

It is a truism of literature that certain authors have a highly recognizable style. The concept of style underlies the authorship attribution techniques that have been applied to tasks such as identifying which of several authors wrote a particular news article. In this paper, we explore whether the works of authors of classic literature can be correctly identified with either of two approaches to attribution, using a collection of 634 texts by 55 authors. Our results show that these methods can be highly accurate, with errors primarily for authors where it might be argued that style is lacking. And did Marlowe write the works of Shakespeare? Our preliminary evidence suggests not.

Keywords: Stylistics, authorship search, language modelling, document management

1 Persuasion

The notion of style is central to literature. The best-known authors of classic English novels and plays are renowned for having distinctive styles that make their works immediately recognizable. From the complexities of Henry James, the humour of Dickens, and the directness of Austen to the folksiness of Twain and the simplicity of London, a reader who is familiar with particular novelists can easily recognize their writing. Some authors are read as much for their style and writing as for what they have to say.

Style is not easy to define or identify. However, it is on the notion of style that the task of authorship attribution (AA) depends: that some element of an author's writing can be used as a reliable marker of their work. Given such markers, AA techniques can be used to verify whether a particular work is by a particular author, or to identify a likely author from amongst a set of candidates. Applications include forensics, plagiarism detection, and analysis of literature.

Current AA techniques have two components: an indexing mechanism for extracting style markers from text, and a comparison mechanism for using the markers to determine probable authorship. The style markers that have been used for this task have been relatively limited; in most work the markers have been distributions of text elements such as function words (Burrows 1987, Binongo 2003, Baayen et al. 2002, Juola & Baayen 2003, Holmes et al. 2001, Zhao & Zobel 2005), punctuation symbols (Baayen et al. 2002),

and part-of-speech tags (Kukushkina et al. 2000, Stamatatos et al. 1999, 2001, Baayen et al. 1996, Zhao et al. 2006).

Comparison mechanisms have been more diverse. Most of the methods are based on statistical analysis, such as principle component analysis (PCA) (Baayen et al. 1996, Holmes et al. 2001, Burrows 2002), linear discriminant analysis (Baayen et al. 2002, Stamatatos et al. 2001); and machine learning techniques, such as Bayesian networks and support vector machines (SVMs) (Diederich et al. 2003, Koppel & Schler 2004), and treat AA as a classification problem. Some work is driven by particular AA problems that researchers have chosen to investigate, and most investigations have involved small volumes of data and small numbers of authors, such as the 65 Federalist Papers of known authorship, each written by one of two people (Fung 2003, Khmelev & Tweedie 2002).

In previous work we have investigated several aspects of AA, including style markers and comparison methods (Zhao & Zobel 2005, Zhao et al. 2006). We have found, in agreement with other researchers, that function words are a reliable indicator of authorship. Using newswire data, we have explored several AA methods, finding that the best results are yielded by SVMs and statistical methods based on language models and entropy (Zhao et al. 2006). Methods such as SVMs can be effective, but are not efficient; it is far from obvious that they can be scaled to the data collections found in typical text repositories. In work in progress, we have explored a search-based method of AA, in which language models are used to match documents by style rather than by content (Zhao & Zobel n.d.). This approach—also tested on newswire data—can be used for a collection of 500,000 documents. However, even though these methods are successful on average, they are not successful in some particular cases. Why this occurs has been unclear.

In this paper, to further explore the properties of AA methods, we apply them to a corpus of novels extracted from the Gutenberg project. While not a large corpus by text collection standards, it is more substantial than the collections used in most previous work for AA, and contains a substantial cross-section of 19th-century English literature as well as other work. Using this collection, we explore the use of three types of style markers—function words, part-of-speech tags (POS), and POS pairs—and the combination of these.

Our results show that authorship can be attributed with high reliability, with classification substantially outperforming search-based AA, while function words are more effective than other types of style markers. Overall results for the best method, using complete texts as queries, give attribution accuracy on positive examples of over 85% and on negative examples of over 95%. Use of parts of texts was less

successful, with 1000-word fragments only achieving 10% overall accuracy, while classification on 10,000-word fragments achieved 53% accuracy—not as good as with complete documents, but sufficient to give an indication of likely authorship.

The errors, that is, cases where texts are misattributed, are illuminating. The commonest error is to misattribute a text that was not originally written in English, suggesting that style—as measured by our methods—does not survive the translation process. A difficult author was Wilde, a result that is perhaps unsurprising given that he was a satirist and thus an imitator of other people's styles. Another difficult author was Defoe, the earliest novelist in our collection. Other errors were not so easily classified, but an interesting (though weak) trend was to misattribute to another author from the same period. Overall, these errors show that the problems in AA probably lie as much in the work as in the method: when given texts that are expected to have identifiable style, AA appears to be highly reliable.

Revisiting a well-known AA problem, in our collection we included plays by several major playwrights of the late sixteenth and early seventeenth century: Marlowe, Jonson, Beaumont & Fletcher (who wrote together), and Shakespeare. In the rare cases that these works were misattributed by the methods we used, they were attributed to another of the group, not unsurprisingly given the changes in English between these works and those of the later authors that made up the bulk of our corpus. However, if Marlowe wrote Shakespeare, as has sometimes been speculated, there is little evidence for it here.

2 A Study in Stylistics

Authorship attribution (AA) is the process of attempting to identify the likely authorship of a given document, given a collection of documents whose authorship is known. Most of the methods described in the research literature consist of two components, an indexing mechanism and a comparison mechanism. The indexer converts each document to a set of tokens or markers whose properties are assumed to be characteristic in some way of a particular author. The comparator uses these markers to assign an author to unattributed documents.

Authorship attribution is an example of use of stylistic aspects of text in retrieval (Pol 2005, Sarkar et al. 2005, Kaster et al. 2005). Style concerns the way in which a document is written rather than its contents; stylistics is the study of style. Automated analysis of stylistics can be applied to a range of problems, from document attribution and authentication to matching document readability to the abilities of the user.

All published AA methods make use of collections of training data of known attribution. These are used to establish models of one kind or another. One data collection is the 65 Federalist papers, written during the debate that led to the creation of the US constitution. As another example of this kind, Holmes et al. (2001) used a collection of 17 journal articles by two authors. Other researchers have used data collections developed specifically for AA research. For example, Baayen et al. (2002) asked eight students to write a total of 72 articles. In our previous work (Zhao & Zobel 2005), we used attributed articles drawn from newswire data, focussing on authors who had made several hundred contributions each. Our motivation was to have a data collection of realistic difficulty both in size and kind. As authors of news articles aim primarily to communicate rather than create art, and as news articles are typically no more than a few thou-

sand words long, we felt that success on such data would be more significant than on some other data sets that have been used. In work in progress we report on successful AA on a collection of 500,000 newswire articles (Zhao & Zobel n.d.).

These datasets are used to test AA in different ways. Some AA problems are two-class, that is, the collection and the unknown documents are all by one of two known authors. Some AA problems are multi-class, which is the generalization of two-class to multiple authors. Some AA problems are one-class, in which some of the documents are by a given author and the rest are unknown; the task is to identify whether a new document is or is not by the given author.

Within computer science, the focus of research has been on comparators rather than indexers, with most researchers assuming a straightforward indexing method and using it as input to a comparator. A common indexing method is to extract function words (or closed-class words) (Burrows 1987, Baayen et al. 2002, Juola & Baayen 2003, Holmes et al. 2001, Kukushkina et al. 2000, Binongo 2003). An alternative is to use NLP methods to annotate the text with parts-of-speech (POS) tags (Baayen et al. 1996, Kukushkina et al. 2000, Stamatatos et al. 1999, 2001, Li et al. 2006, Masuyama & Nakagawa 2004), and to use these tags—or sequences of tags—as markers. Use of POS is intuitively attractive, but work to date has found POS tags to be no more effective than function words, a result that is confirmed in this paper.

Attribution is a form of classification, and thus it is attractive to apply existing classification methods; many of the proposed AA methods are based on classification techniques. Classification has been investigated in areas such as machine learning (Scholkopf & Smola 2002, Witten & Frank 2000, Quinlan 1993), text categorization (Bekkerman et al. 2003, Gabrilovich & Markovitch 2004, Khmelev & Teahan 2003, Li et al. 2003), and speech recognition. For AA, a range of classification-based attribution methods have been proposed. Binary authorship attribution is the simplest case. Binongo (2003) used PCA to investigate the writing pattern of the fifteenth book of *Oz*, as a case study of binary classification. In another case study, Holmes et al. (2001) also used PCA to distinguish between two authors. Multi-class and one-class AA are considered to be harder problems. Diederich et al. (2003) applied SVM for multi-classification AA and reported accuracy from 60% to 80%. Fung (2003) used SVM for feature selection on the Federalist papers. Koppel & Schler (2004) proposed an “unmask” approach for one-class AA, based on case-by-case selection of individual features for each author, and achieved around 80% accuracy. All words, not just function words, were considered.

Some methods have made use of the full text of documents (Diederich et al. 2003, Benedetto et al. 2002) rather than markers such as function words. Despite claimed good results, the plausibility of these methods is questionable, as they are based on word-occurrence statistics and choice of words is then as much a product of topic as of style; in one reported case an attempt to reproduce the results failed (Goodman 2002), and in unreported experiments we found that attribution based on full text of newswire articles was a complete failure.

A difficulty in examining much of this past work is that the methods were tested on different collections, making comparison of results far from straightforward. We compared a selection of these methods on our newswire collection (Zhao & Zobel 2005, Zhao et al. 2006) and found that Bayesian networks and SVMs were the most effective. However, there are challenges in scaling these methods to large volumes

of data. We therefore investigated an alternative that has lower costs, as we now explain.

3 Priors and Prejudice

Classification methods such as those based on machine learning techniques make use of statistical properties of the items being classified. In computing, one of the most fundamental statistical properties is entropy. Intuitively, it seems plausible that the distribution of features in a new document should approximately match that of other documents by the same author. That is, we could build a model for each author based on known documents, and the model that is most like that of a new document can be assumed to identify its author.

On this basis, in previous work we have proposed an AA method using Kullback-Leibler Divergence (KLD)—a relative entropy measurement (Zhao et al. 2006). The underlying model of language is like that used in information retrieval (IR) (Lafferty & Zhai 2001, Zhai & Lafferty 2001, 2002, 2004). In this approach, the KLD between two models can be measured as:

$$KLD(p||q) = \sum_{x \in X} p(x) \log_2 \frac{p(x)}{q(x)} \quad (1)$$

where $p(x)$ and $q(x)$ are probability mass functions used to calculate the probability of getting instance x . For AA, p is the model for new document d and q is a model based on known works of each author. AA involves computing $KLD(p||q)$ for p and every author model q , and choosing the q that gives the smallest relative entropy.

A simple form of model is the maximum likelihood $p(x) = f_{x,d}/|d|$, where $f_{x,d}$ is the number of occurrences of feature x in d and $|d|$ is the total number of feature occurrences in d . However, this has drawbacks in principle and in practice. Regarding these models as generative, the features that are observed in a document are a subset of those that might have occurred, and their absence from a document leads to obvious computational difficulties. Also, there is no control for the frequency of each feature in language in general. For example, if the features are function words, we would expect occurrences of a word such as “whilst” to be moderately indicative of authorship, while occurrences of a word such as “the” are uninformative. For this reason we need to use smoothing (Chen & Goodman 1996, Hiemstra 2002, Zhai & Lafferty 2001, 2004).

In our work, the probability mass function is formulated with Dirichlet smoothing, which is to date one of the most effective smoothing techniques in IR (Zhai & Lafferty 2004):

$$p'_d(x) = \frac{|d|}{\mu + |d|} p(x) + \frac{\mu}{\mu + |d|} p_B(x) \quad (2)$$

Here, $p_B(x)$ is the probability of component x in a *background model*, which is used to address the problem caused by zero probabilities and to provide global feature statistics. The background model can be viewed as a collection of prior probabilities that can be used to bias the KLD to favour less common features. The parameter μ adjusts the significance of documents and background model contributing to the term weights. This parameter must be tuned; if it is too low, there is little discrimination between features, but if it is too high, the statistical properties of the document may be obscured. The value of μ can be critical for short documents where the statistical evidence is noisy; for long documents, which

predominate in the experiments reported here, the exact value of μ is relatively unimportant. The background model should ideally be drawn from a large collection of independent text.

Combining Equations 1 and 2, the dissimilarity $KLD(p_d||p_q)$ between two sets of features d and q can be written as:

$$\sum_{x \in q \cup d} \left[\left(\frac{f_{x,d}}{\mu + |d|} + \frac{\mu}{\mu + |d|} p_B(x) \right) \times \log \frac{\frac{f_{x,d}}{\mu + |d|} + \frac{\mu}{\mu + |d|} p_B(x)}{\frac{f_{x,q}}{\mu + |q|} + \frac{\mu}{\mu + |q|} p_B(x)} \right]$$

In the context of AA, we argue, use of KLD provides a principled approach where classification, rather than being based on elaborate statistical methods, is directly derived from simple fundamental theory (Zhao et al. 2006). Our results show that KLD is as effective as the best competitor method, SVMs, for binary AA, and is also effective for multi-class AA, a task for which SVMs are unsuited. In contrast to SVMs, the training cost is small, consisting only of counting of features.

Moreover, KLD-based AA can plausibly be applied to large collections. In work in progress (Zhao & Zobel n.d.), we have further explored relative entropy, but instead of finding the model that is most similar we use KLD as a way of ranking the documents in a collection according to the similarity of their feature sets to that of a query. This is, in principle, little different to standard text retrieval with a search engine (Zobel & Moffat 2006), and provides several potential advantages. The heuristics used during standard search can be applied, allowing rapid identification of the most similar documents (or rather, sets of features that represent documents) extremely fast. If the top-ranked documents are consistently by a given author, there is a clear indication of authorship of the query document; if, however, the highly-ranked documents are mixed, then it is likely that attribution is uncertainty. (An estimate of certainty is a useful guide to the quality of AA results.) Alternatively, given a query of known authorship, the matches are the documents most likely to be by the same author, an approach that has promise in for example plagiarism detection.

In work in progress (Zhao & Zobel n.d.), we have found that the KLD ranking approach is effective for authorship search. The largest collection we are using contains 500,000 documents, in which only 100 are positive examples by an author. The baseline probability of finding a single correct match in the top 10 is only 0.2%. We obtained an average of up to 44% of matches being correct in the top 10 documents in the ranking. It is therefore interesting to explore whether search works as an attribution method on other collections, as in the experiments described below. It is also interesting to explore how search compares to classification.

In this paper, our focus is on the problem of attribution in English literature. Are simple style markers sufficient for accurate identification of the author of a work? When attribution fails, what is the cause? Our experiments, described below, explore these questions.

4 Tests of the Diverse Styles

Our aims in this paper are to compare the effectiveness of search and classification as attribution methods; to see how effective attribution is on literature; and to understand when and why attribution fails. Given that we plan to use KLD for attribution, the

other elements that need to be established are an indexing method and a testbed dataset. We describe these below, then report on our experiments.

Sets and Sensitivity

A key aspect of this investigation was to explore the effectiveness of attribution on literature. The task may be relatively easy if the collection is small or there are only a few authors, or if the authors are from widely different periods. We sought to collect literature that was representative and consistent. Using the Gutenberg collection,¹ we gathered books from about 50 of the top-100 most downloaded authors. In most cases we collected 10 books, or fewer if less than 10 were available, but in some cases collected all works. The total number of books collected was 634, and the total number of authors (including playwrights as discussed below) was 55. We call this collection Gutenberg634.

In selecting the books, we avoided choices that we felt were inconsistent with the aims of our experiments. We did not collect volumes of poetry, dictionaries, or text in languages other than English. Individual short stories were avoided, especially in cases where a collection containing the story was also available. Authors with four or fewer works were not considered.

However, we did keep both plays and novels; plays were greatly in the minority but allowed us to examine attribution of the works of playwrights from the time of Shakespeare. The complete list of authors is shown in Table 2.

Maintenance of consistency was not straightforward. One book may have many different editions, or may be presented in different forms, such as both a complete edition and as a series of parts. The raw documents contain other-author material such as the Gutenberg disclaimer, editors' commentaries, and sometimes an introduction and preface written by someone else. We ensured in most cases that the major works of the author are included and that no duplicates are included. Finally, we manually deleted all other-author text from each book.

In the experiments, we built 634 collections of 633 documents each; that is, in each case one of the documents was left out to be used as a query (in other words, to be classified). In attribution, we can simply count the classification accuracy, based on N_c , how many documents were correctly classified. In search, other measures are possible. We use N_r , the number of documents by the same author ranked in the top 5. Given the maximum possible value for N_c (respectively, N_r), we can then give a percentage accuracy for each technique. The number of correct documents in the top 5, or precision at 5 documents returned, we denote as P@5. As can be seen in the tables, results are broken down by author and by method.

Measure for Measure

Indexing was one of the aspects of AA explored in our experiments. We tested several different simple forms of marker types. One form of marker was function words, which are content-free but grammatically important words such as prepositions, conjunctions, articles, and elements such as words describing quantities. Function words have shown to be reasonably effective for binary and multi-class AA in our previous work (Zhao & Zobel 2005, Zhao et al. 2006, Zhao & Zobel n.d.). We used a list of 363 function words.

Another form of marker was POS tags. We tagged the entire Gutenberg634 collection using NLTK (a

Table 1: *Usage statistics for the commonly used style markers for two authors. Each number is, for that author, the percentage of function word occurrences that is the particular function word. Counts are averaged across all documents available by each author.*

	Function words			POS tags		
	the	of	a	cc	in	jj
Shakespeare	7.6	4.8	4.1	3.8	5.9	2.8
Marlowe	9.5	6.2	3.2	3.2	6.4	2.4

natural language toolkit).² We used all tagged works of fiction from the Brown corpus³ to train a unigram tagger. The accuracy of the trained tagger is 86.73%. In addition, we trained a Bigram tagger, but accuracy was only 83.23%, and we used the unigram tagger in our experiment. A list of 183 POS distinct tags was used. For example, for the text

The widow she cried over me, and called me
a poor lost lamb, and she called me a lot of
other names, too, but she never meant no
harm by it.

the function words extracted were “the over and a and a of other too but never no by it”. The POS tags were “at nn pps vbd in ppo cc vbd ppo at jj vbn zz cc pps vbd ppo at nn in ap nns ql cc pps rb vbd at nn in ppo”, in which, for example, nn is a noun.

We used the POS tags both individually and as pairs. The pairs should in principle give an indication of the way in which the author combines parts of speech, which is plausibly a signature of the author's style. However, automatic identification of POS may to some extent undermine this aim, as POS tagging is most likely to fail when presented with atypical word sequences and sentence formations—the very aspects of text that characterize style. That is, POS tags are likely to be least reliable for the most interesting elements in the text. In our results we also show the effect of combining the evidence of the different marker types, and thus have four sets of results in each of the two main experiments.

Table 1 gives an example of how usage of different type of style markers can vary between authors. In this example from the collection of Gutenberg634, for even common style markers, the usage can be quite different.

As noted earlier, the smoothing parameter μ in Equation 2 plays an important role. We observed that, setting $\mu = 1000\sqrt{10}$ has given the best performance in our previous work (Zhao et al. 2006), in which we test binary AA with a small dataset of chapters of novels derived from the Gutenberg collection. (Chapters were extracted manually, an infeasible process with the larger Gutenberg634.) We used this value for parameter μ in our experiments here.

Choice of background model is another important factor in such experiments. Based on our experience with KLD in IR and AA, we believe that 634 books are not sufficient for a good background model. Therefore, we collected the background model from the AP collection, of over 250,000 newswire articles; AP is a sub-collection of the TREC data.⁴ In some respects this choice is not ideal, consisting as it does of non-fiction written in the late 1980s and early 1990s, but was the best option available to us. This text

²Available from nltk.sourceforge.net/index.html.

³The Brown corpus consists of one million English words gathered in 1961. The texts for the corpus are grouped into fifteen text categories. The corpus is the first of the modern computer readable corpora.

⁴See trec.nist.gov.

¹See www.gutenberg.org.

had been fully tagged as part of our earlier experiments (Zhao & Zobel n.d.). As the results show, AA is highly effective with this background model; a better background model may further improve results, but they are already strong.

Great Expectations

Attribution via search provides a way of identifying the authorship of a new document, and of finding other documents that the author has written. Our aim in the first experiment was to test the effectiveness of different style markers when used in search-based AA. That is, the set of style markers extracted from each book is used as a query, and the other books are ranked according to their similarity. The hypothesis is that, if the markers are a good indication of style, then the highest documents in the ranking should be by the same author.

Each book was indexed in three ways: by function words, by POS tags, and by POS-tag pairs. For each form of indexing, we had 634 runs; in each, one of the books was used as a query and the remainder of the Gutenberg634 collection as a corpus. For each book, the rankings from the different forms of indexing were combined to give a fourth set of results. Performance for each query was measured with P@5, that is, the number of works by the same author in the top five ranked results. Outcomes, by author, are shown in Table 2. The “optimal retrieval” column shows the maximum number of correct results that could be obtained for each author. For example, Curtis wrote 7 books; the optimal result was 35; using function words, only 19 (or 54.3%) were retrieved; while results for Curtis using other style markers were somewhat lower.

As these results show, KLD-based search on markers is an effective mechanism for matching texts by authorship. Using function words as markers, on average over 76% of the documents in the top 5 are a correct match, and for 15 of the 55 authors accuracy is 90% or better. Other markers are somewhat less successful, but still reasonably effective, with 62% for POS tags and 66.5% for POS-tag pairs. Combination does no better than the average of the techniques being combined, at 71%. We had hoped that POS tags would prove the more effective method; and hypothesize, following the earlier discussion, that the very qualities that make an author’s style unique may lead to tagging failures.

However, search-based AA is not particularly successful for some authors. An elementary cause might be the number of training examples; results were somewhat better for authors with more texts.

Other causes are attributable to style. Consider Schiller (German) and Tolstoy (Russian), two of the four authors whose works were originally written in a language other than English; the other two are Maupassant and Verne, both of whom originally wrote in French. Schiller and Tolstoy are amongst the worst cases for search-based AA with function words. Presumably the process of translation, or the fact that multiple translators may be involved, obviates some of the individuality of style.

An interesting element in the errors is that there was a weak tendency for mismatches to be in the right period. For example, as discussed further below, when the works of Marlowe were used as queries, most of the matches were plays written by his peers.

Finally, to state the somewhat obvious, some authors do not have a strong writing style that can be easily identified easily, and other authors change their style between books. It is perhaps not surprising that the works of Wilde and Bierce, both satirists, prove difficult to attribute.

As another perspective on these results, we reassessed the quality of attribution using the following rules. Given a query text by some author A , if three or more of the top 5 matches were by some author A' , then we attributed the query text to A' with high confidence (and were right if $A = A'$). If two of the top 5 were by A' and the remainder were by three different authors, then we attributed the query text to A' with low confidence. Otherwise we judged the attribution to be unknown.

For function words, we attributed with strong confidence correctly in 451 cases and incorrectly in 61 cases, an accuracy of 88%. We attributed with low confidence correctly in 20 cases and incorrectly in 23 cases, not a wonderful result but much better than random. Attribution was unknown in 79 cases. Overall accuracy was 74%. It is against this result that a classification-based attribution method should be compared.

Through The Looking-Class

In our next experiment, we used KLD for one-class AA, on the same sets of markers. We again had 634 runs for each kind of markers. In each run, the aim was to make a decision on authorship—whether the query text was by a given author or was more likely to be by someone else. For example, for Austen we created a positive model using seven of her texts, created a negative model using the 626 texts by other authors, and used Austen’s remaining text as a query. This was repeated for each of Austen’s eight texts, and then for every other author. That is, classification was on a positive leave-one-out approach.

Results are shown in Table 3. As can be seen, classification is rather more effective than search-based attribution, achieving, for function words, an average of over 85%, in contrast to 74% for search-based attribution. Better than 90% accuracy is observed for 30 of the 55 authors. The POS-tag methods have proved much more successful than previously, but effectiveness is still slightly lower than with function words. Combination yielded little benefit. Improvement with POS tags requires, we believe, a more robust tagging method.

Similar failures can be observed. Schiller and Tolstoy are again problematic, as is Wilde. Another difficult author is Defoe, perhaps surprisingly, as he is the only author from the early eighteenth century. Overall, however, the results are highly satisfying.

Positive classification results give an estimate of the rate of false misses. Negative classification results are required to estimate the rate of false matches. That is, for a model trained on some author, say Austen, and a work by some other author, say Alcott, we wish to know the likelihood that the work will be attributed as by Austen. In these experiments, correctness is 95.2%, much higher than the accuracy on positive examples.

The texts we have used in these experiments are complete books, averaging over 80,000 words each. A question then is whether AA would be accurate on smaller texts. We re-ran the positive leave-one-out experiments using the full texts for training and, for each text being tested, a single 1000-word fragment as a query. Each fragment was drawn from a few thousand words after the start of the text. These experiments were not successful, with overall accuracy of only 10.4%. Use of 10,000-word fragments was more effective, giving overall accuracy of 53.2%. This result is far from perfect, but is much better than random, where average accuracy of around 3% would be expected. At this level accuracy, AA is not conclusive, but is nonetheless highly indicative.

Table 2: Results of authorship search experiments. Function words, POS tags, POS pairs, and combined features are used as style markers. Results are total P@5 per author and a percentage of optimal retrieval.

# of books per author	Optimal retrieval	Function words		POS tags		POS pairs		Mixed	
		N_r	$P@5$	N_r	$P@5$	N_r	$P@5$	N_r	$P@5$
Alcott(10)	50	43	86.0	32	64.0	32	64.0	38	76.0
Alger(10)	50	50	100.0	47	94.0	50	100.0	50	100.0
Austen(8)	40	39	97.5	31	77.5	37	92.5	38	95.0
Baum(10)	50	45	90.0	42	84.0	44	88.0	44	88.0
Bierce(8)	40	6	15.0	4	10.0	5	12.5	4	10.0
Burroughs(9)	45	39	86.7	21	46.7	27	60.0	33	73.3
Carroll(6)	30	7	23.3	4	13.3	1	3.3	3	10.0
Churchill(22)	110	93	84.5	75	68.2	78	70.9	87	79.1
Collins(23)	115	105	91.3	94	81.7	101	87.8	103	89.6
Conrad(12)	60	51	85.0	24	40.0	32	53.3	39	65.0
Curtis(7)	35	19	54.3	9	25.7	12	34.3	14	40.0
Darwin(9)	45	28	62.2	31	68.9	29	64.4	31	68.9
Defoe(9)	45	22	48.9	20	44.4	19	42.2	21	46.7
Dickens(11)	55	40	72.7	11	20.0	16	29.1	16	29.1
Fletcher(6)	30	23	76.7	19	63.3	20	66.7	22	73.3
Galsworthy(10)	50	22	44.0	27	54.0	29	58.0	30	60.0
Haggard(37)	185	168	90.8	154	83.2	165	89.2	168	90.8
Hardy(7)	35	35	100.0	18	51.4	15	42.9	18	51.4
Harte(9)	45	36	80.0	36	80.0	37	82.2	41	91.1
Hawthorne(10)	50	30	60.0	31	62.0	32	64.0	37	74.0
Henry(9)	45	40	88.9	37	82.2	40	88.9	40	88.9
Holmes(9)	45	30	66.7	20	44.4	20	44.4	25	55.6
Howells(10)	50	23	46.0	15	30.0	20	40.0	23	46.0
James(19)	95	80	84.2	48	50.5	44	46.3	58	61.1
Jonson(7)	35	19	54.3	26	74.3	30	85.7	29	82.9
Kingsley(10)	50	28	56.0	17	34.0	13	26.0	20	40.0
Kipling(8)	40	28	70.0	12	30.0	19	47.5	19	47.5
Lang(10)	50	19	38.0	11	22.0	14	28.0	14	28.0
Lever(9)	45	40	88.9	40	88.9	38	84.4	40	88.9
London(21)	105	101	96.2	64	61.0	79	75.2	93	88.6
Lytton(10)	50	49	98.0	49	98.0	43	86.0	49	98.0
MacDonald(9)	45	26	57.8	11	24.4	18	40.0	19	42.2
Marlowe(5)	20	6	35.0	6	30.0	8	40.0	9	45.0
Maupassant(9)	45	40	88.9	33	73.3	37	82.2	36	80.0
McCutcheon(10)	50	45	90.0	35	70.0	45	90.0	50	100.0
Motley(10)	50	50	100.0	50	100.0	45	90.0	50	100.0
Parker(10)	50	40	80.0	33	66.0	21	42.0	34	68.0
Pepy(10)	50	50	100.0	50	100.0	50	100.0	50	100.0
Poe(6)	30	21	70.0	18	60.0	19	63.3	22	73.3
Rohmer(10)	50	50	100.0	46	92.0	48	96.0	50	100.0
Schiller(10)	50	19	38.0	21	42.0	22	44.0	21	42.0
Scott(10)	50	50	100.0	49	98.0	50	100.0	50	100.0
Shakespeare(42)	210	203	96.7	197	93.8	199	94.8	201	95.7
Shaw(10)	50	33	66.0	28	56.0	30	60.0	30	60.0
Stevenson(10)	50	11	22.0	4	8.0	9	18.0	8	16.0
Stockton(10)	50	38	76.0	23	46.0	33	66.0	33	66.0
Tolstoy(15)	75	38	50.7	26	34.7	28	37.3	30	40.0
Twain(14)	70	57	81.4	33	47.1	46	65.7	50	71.4
Verne(10)	50	41	82.0	35	70.0	46	92.0	45	90.0
Wake(9)	45	34	75.6	40	88.9	38	84.4	40	88.9
Warner(10)	50	27	54.0	26	52.0	28	56.0	29	58.0
Wells(10)	50	23	46.0	17	34.0	23	46.0	22	44.0
Wilde(7)	35	2	5.7	2	5.7	1	2.9	2	5.7
Wodehouse(23)	115	113	98.3	97	84.3	100	87.0	102	88.7
Yonge(10)	50	33	66.0	13	26.0	21	42.0	21	42.0
Total(634)	3165	2409	76.1	1962	62.0	2106	66.5	2251	71.1

Table 3: Results of one-class authorship attribution. Function words, POS tags, POS pairs, and combined features are used as style markers. Results are total correct per author and a percentage of correct attribution.

Author Name	# of book in total	Function Words		POS tags		POS pairs		Mixed	
		N_c	Accuracy	N_c	Accuracy	N_c	Accuracy	N_c	Accuracy
Alcott	10	9	90.0	9	90.0	8	80.0	9	90.0
Alger	10	10	100.0	10	100.0	10	100.0	10	100.0
Austen	8	8	100.0	7	87.5	7	87.5	7	87.5
Baum	10	10	100.0	9	90.0	9	90.0	9	90.0
Bierce	8	6	75.0	6	75.0	5	62.5	5	62.5
Burroughs	9	8	88.9	8	88.9	8	88.9	8	88.9
Carroll	6	3	50.0	3	50.0	2	33.3	2	33.3
Churchill	22	20	90.9	19	86.4	18	81.8	18	81.8
Collins	23	21	91.3	18	78.3	19	82.6	19	82.6
Conrad	12	12	100.0	11	91.7	11	91.7	12	100.0
Curtis	7	6	85.7	5	71.4	5	71.4	5	71.4
Darwin	9	6	66.7	6	66.7	7	77.8	7	77.8
Defoe	9	5	55.6	5	55.6	5	55.6	5	55.6
Dickens	11	8	72.7	6	54.5	6	54.5	6	54.5
Fletcher	6	6	100.0	6	100.0	6	100.0	6	100.0
Galsworthy	10	8	80.0	5	50.0	5	50.0	5	50.0
Haggard	37	26	70.3	31	83.8	30	81.1	30	81.1
Hardy	7	7	100.0	3	42.9	6	85.7	6	85.7
Harte	9	8	88.9	9	100.0	9	100.0	9	100.0
Hawthorne	10	5	50.0	9	90.0	8	80.0	8	80.0
Henry	9	9	100.0	9	100.0	9	100.0	9	100.0
Holmes	9	9	100.0	9	100.0	8	88.9	9	100.0
Howells	10	6	60.0	6	60.0	6	60.0	6	60.0
James	19	17	89.5	17	89.5	17	89.5	17	89.5
Jonson	7	7	100.0	7	100.0	7	100.0	7	100.0
Kingsley	10	9	90.0	9	90.0	9	90.0	9	90.0
Kipling	8	8	100.0	7	87.5	7	87.5	7	87.5
Lang	10	7	70.0	2	20.0	4	40.0	4	40.0
Lever	9	8	88.9	4	44.4	8	88.9	8	88.9
London	21	21	100.0	21	100.0	20	95.2	20	95.2
Lytton	10	9	90.0	10	100.0	10	100.0	10	100.0
MacDonald	9	7	77.8	5	55.6	7	77.8	6	66.7
Marlowe	5	5	100.0	5	100.0	5	100.0	5	100.0
Maupassant	9	7	77.8	8	88.9	7	77.8	8	88.9
McCutcheon	10	10	100.0	10	100.0	10	100.0	10	100.0
Motley	10	10	100.0	10	100.0	10	100.0	10	100.0
Parker	10	8	80.0	10	100.0	8	80.0	8	80.0
Pepy	10	10	100.0	10	100.0	10	100.0	10	100.0
Poe	6	6	100.0	6	100.0	6	100.0	6	100.0
Rohmer	10	10	100.0	10	100.0	10	100.0	10	100.0
Schiller	10	7	70.0	9	90.0	9	90.0	8	80.0
Scott	10	10	100.0	10	100.0	10	100.0	10	100.0
Shakespeare	42	40	95.2	41	97.6	41	97.6	41	97.6
Shaw	10	9	90.0	8	80.0	8	80.0	8	80.0
Stevenson	10	7	70.0	6	60.0	5	50.0	6	60.0
Stockton	10	9	90.0	8	80.0	8	80.0	8	80.0
Tolstoy	15	8	53.3	7	46.7	6	40.0	7	46.7
Twain	14	13	92.9	12	85.7	12	85.7	13	92.9
Verne	10	10	100.0	10	100.0	10	100.0	10	100.0
Wake	9	6	66.7	9	100.0	9	100.0	9	100.0
Warner	10	8	80.0	9	90.0	9	90.0	9	90.0
Wells	10	9	90.0	8	80.0	8	80.0	8	80.0
Wilde	7	2	28.6	3	42.9	2	28.6	2	28.6
Wodehouse	23	22	95.7	20	87.0	21	91.3	21	91.3
Yonge	10	8	80.0	7	70.0	8	80.0	9	90.0
Total	634	543	85.6	527	83.1	528	83.3	534	84.2

Table 4: Example ranked lists (top 5) for five works of Shakespeare; markers are function words only.

Rank	Sh139	Sh149	Sh155	Sh163	Sh166
1	Sh166	Sh165	Sh128	Sh162	Sh139
2	Sh145	Sh21	Sh162	Sh166	Sh148
3	Sh148	Sh29	Sh167	Sh169	Sh147
4	Sh147	Sh164	Sh147	Sh23	Sh145
5	Sh155	Sh22	Sh164	Sh168	Sh155

Master and Man

The hypothesis that the works of Shakespeare were written by someone else has been argued for hundreds of years.⁵ As a preliminary investigation into whether our methods could throw any light on the debate, we included in Gutenberg634 the works of major playwrights of Shakespeare's time: William Shakespeare, Francis Beaumont & John Fletcher (whose works are co-authored), Ben Jonson, and Christopher Marlowe. By examining the extent to which these works were consistent with each other, and whose works matched to whose, we speculated that we might discover some evidence pointing in one direction or the other.

An admitted weakness of such an investigation is that our tools are not particularly sophisticated. These texts have been subjected to intensive literary analysis for several centuries and it would be foolhardy of us to suppose that a straightforward statistical analysis would lead to dramatic revelations. Nonetheless, it is our belief that patterns of writing are not easily mimicked or disguised. Should an author be pretending to be Shakespeare, for example, we would hope to observe inconsistencies in the statistical character of Shakespeare's works.

Another caveat is that we have not spread our net wide. Many candidates have been proposed as the authors of the works of Shakespeare; however, of these, only the works of Marlowe were available to us in a suitable form.

To examine this question of authorship, consider first the search experiments reported earlier, in which function words were used as markers. We now examine the ranked lists for selected books by each of these four cases, Shakespeare, Beaumont & Fletcher, Marlowe, and Jonson. For simplicity, we use a shorthand to indicate the writers in the following tables: "Sh", "BF", "Ma", and "Jn". Each notation is followed by a number, derived from filenames in Gutenberg634, so that for example Sh165 represents *Timon of Athens*.

In Table 4, we have listed the authorship of the top 5 retrieved books for each of five of Shakespeare's texts. For Shakespeare, we found an extremely high consistency of writing, with, overall, 203 of 210 top-5 listings being correct.

In Tables 5, 6, and 7 we show the ranked lists for Beaumont & Fletcher Marlowe, and Jonson. These results are rather less consistent than for Shakespeare, perhaps unsurprisingly given that there are far fewer training texts for these authors.

The best case is that of Beaumont & Fletcher, where only six of the 25 documents are mismatches. The cases of Marlowe and Jonson are more intriguing. Marlowe's rankings are dominated by the works of Shakespeare, with 17 of the 25 matches. Jonson is hardly better, with Shakespeare giving 14 of the 25 matches. In both cases the actual works of the author

Table 5: Example ranked lists for works of Beaumont & Fletcher; markers are function words only.

Rank	BF19	BF20	BF21	BF22	BF23
1	BF24	BF21	BF20	BF21	BF20
2	BF23	BF23	BF22	BF20	BF24
3	Sh149	BF22	BF23	BF23	BF21
4	Sh165	BF24	BF24	BF24	BF22
5	Sh159	Jn7	Jn8	BF19	Jn8

Table 6: Example ranked lists (top 5) for works of Marlowe; markers are function words only.

Rank	Ma11	Ma12	Ma13	Ma14	Ma17
1	Sh166	Ma13	Ma14	Ma13	Sh166
2	Sh163	Sh139	Sh139	Sh139	Sh139
3	Sh148	Ma14	Ma12	Ma12	Sh147
4	Sh139	Ma17	Sh166	Sh166	Sh155
5	Sh169	other	Sh147	Sh147	Sh148

are not prominent. Note too that all but one of the 100 matches is by a playwright of this era—they may be conflated with each other, but there is no doubting when these plays were written.

So, does the evidence suggest that Marlowe wrote Shakespeare?

The circumstances of Marlowe's death, whether in a tavern or in an assassination, have been debated for longer than the question of authorship of the works of Shakespeare. Some people argue that Marlowe faked his death and used "Shakespeare" as his pen name to continue writing afterwards. However, our results do not suggest a particular relationship between the works of Marlowe and Shakespeare.

It is true that plays by Marlowe tend to retrieve plays by Shakespeare, as seen in Table 6. However, the evidence becomes weaker when we compare Table 4 and Table 6 in detail. Sh139 appears five times in five searches in Table 6. Given the hypothesis that the true author for this book is Marlowe, it should occasionally retrieve books by Marlowe. However, as can be seen in Table 4, when Sh139 is used as a query, no works of Marlowe are retrieved. Sh166 and Sh147 share the same properties—none of these retrieve books by Marlowe. The fact that Jonson's works also match those of Shakespeare further suggests that the similarity with Marlowe may be a matter of period rather than authorship.

The positive leave-one-out experiments are also indicative. In these experiments, the plays of Marlowe and Jonson are never misattributed. To some extent this may be due to experimental design—the presence of Shakespeare's plays in the negative examples is watered down by the great volume of nineteenth-century text. However, in the negative leave-one-out experiments, the works of both Marlowe and Jonson are usually attributed to Shakespeare, while those of Beaumont & Fletcher are occasionally attributed to Shakespeare. That is, the rate of false matches is extremely high, and the works of these authors cannot be distinguished. At the same time, there is no particular evidence that the works of any of these authors has unusually high similarity to that attributed to Shakespeare. Taking these considerations together we see no evidence in our experiments to support the hypothesis that Marlowe wrote Shakespeare.

⁵Some say that the proposition was first put by Edward Blount in 1623, others cite Queen Elizabeth I. See for example any number of web pages that are returned for the query "marlowe shakespeare". We hesitate to endorse them but they are certainly entertaining. Results for "shakespeare authorship" are also of interest.

Table 7: Example ranked lists (top 5) for works of Jonson; markers are function words only.

Rank	Jn1	Jn5	Jn7	Jn8	Jn9
1	Jn8	Jn7	Jn5	Sh142	Jn8
2	Sh162	Sh168	Jn2	Sh167	Jn2
3	Sh28	Jn1	Sh168	Jn2	Sh147
4	Sh142	Sh8	Jn8	Jn7	Sh139
5	Sh155	Sh156	Sh167	Jn1	Sh26

5 All's Well That Ends Well

We have explored the effectiveness of authorship attribution on works of literature. Using a collection of 634 works derived from the Gutenberg project, our experiments have shown that positive leave-one-out classification can be highly effective, with accuracy of over 85%. Negative leave-one-out experiments, although admittedly incomplete, were even more accurate. Search-based attribution was less successful, but still achieved accuracy of 74%. Not only, then, do these results confirm that authors do indeed have an identifiable writing style, but they confirm that simple markers suffice to identify a particular author.

The best results used function words as markers of style; part-of-speech tags were reasonably effective, but were, we believe, undermined by the fact that tagging is an error-prone process. Tagging tends to fail on text with unusual constructions, and such constructions tend to be indicative of style. In contrast, extraction of function words is straightforward.

Most of our experiments used whole documents as queries. Use of fragments of documents was less successful, with 1000 words being clearly insufficient. Fragments of 10,000 words—somewhat over a tenth of a typical book—allowed correct attribution in over 50% of cases. This result is consistent with our previous exploration of AA on news articles, which are much shorter than books; the accuracy of classification is much better than random, but is insufficient on its own to definitively determine authorship.

These experiments allowed us to examine why attribution sometimes fails. The pattern of errors suggests that a key cause is a lack of distinct style in some texts, such as translated books. That is, some of the failures are due to properties of the works rather than shortcomings of the attribution method. The experiments also allowed us, in a small way, to revisit the question of the authorship of Shakespeare. We did not discover evidence that these works were written by Marlowe.

A limitation of our experiments was that the sources were somewhat mixed, and time prohibited creation of the larger pool of texts that we would have like to have used—each text required some manual editing to remove non-author material. The bulk of the text was from the nineteenth century, but a fraction was much older. Nonetheless, results were highly successful, and provide strong confirmation of the ability of simple statistical methods to accurately identify authorship.

Acknowledgements.

This work was supported by the Australian Research Council.

References

Baayen, H., Halteren, H. V., Neijt, A. & Tweedie, F. (2002), 'An experiment in authorship attribution',

6th JADT.

Baayen, H., Halteren, H. V. & Tweedie, F. (1996), 'Outside the cave of shadows: Using syntactic annotation to enhance authorship attribution', *Literary and Linguistic Computing* **11**(3), 121–132.

Bekkerman, R., El-Yaniv, R., Tishby, N. & Winter, Y. (2003), 'Distributional word clusters vs. words for text categorization', *J. Mach. Learn. Res.* **3**, 1183–1208.

Benedetto, D., Caglioti, E. & Loreto, V. (2002), 'Language trees and zipping', *The American Physical Society* **88**(4).

Binongo, J. N. G. (2003), 'Who wrote the 15th book of Oz? an application of multivariate statistics to authorship attribution', *Computational Linguistics* **16**(2), 9–17.

Burrows, J. (1987), 'Word patterns and story shapes: the statistical analysis of narrative style', *Literary and Linguistic Computing* **2**, 61–70.

Burrows, J. (2002), 'Delta: A measure of stylistic difference and a guide to likely authorship', *Literary and Linguistic Computing* **17**, 267–287.

Chen, S. F. & Goodman, J. (1996), An empirical study of smoothing techniques for language modeling, in A. Joshi & M. Palmer, eds, 'Proc. 34th Annual Meeting of the Association for Computational Linguistics', Morgan Kaufmanns, pp. 310–318.

Diederich, J., Kindermann, J., Leopold, E. & Paass, G. (2003), 'Authorship attribution with support vector machines', *Applied Intelligence* **19**(1-2), 109–123.

Fung, G. (2003), The disputed federalist papers: Svm feature selection via concave minimization, in 'Proc. 2003 Conf. on Diversity in Computing', ACM Press, pp. 42–46.

Gabrilovich, E. & Markovitch, S. (2004), Text categorization with many redundant features: Using aggressive feature selection to make svms competitive with c4.5, in 'Proc. 21st Int. Conf. on Machine learning', ACM Press.

Goodman, J. (2002), 'Extended comment on language trees and zipping'.

Hiemstra, D. (2002), Term-specific smoothing for the language modeling approach to information retrieval: the importance of a query term, in 'Proc. 25th ACM SIGIR Conf. on Research and Development in Information Retrieval', ACM Press, pp. 35–41.

Holmes, D. I., Robertson, M. & Paez, R. (2001), 'Stephen Crane and the New York Tribune: A case study in traditional and non-traditional authorship attribution', *Computers and the Humanities* **35**(3), 315–331.

Juola, P. & Baayen, H. (2003), 'A controlled-corpus experiment in authorship identification by cross-entropy', *Literary and Linguistic Computing*.

Kaster, A., Siersdorfer, S. & Weikum, G. (2005), Combining text and linguistic document representations for authorship attribution, in 'SIGIR workshop: Stylistic Analysis of Text For Information Access'.

- Khmelev, D. V. & Teahan, W. J. (2003), A repetition based measure for verification of text collections and for text categorization, in 'Proc. 26th ACM SIGIR Conf. on Research and Development in Information Retrieval', ACM Press, pp. 104–110.
- Khmelev, D. V. & Tweedie, F. (2002), 'Using markov chains for identification of writers', *Literary and Linguistic Computing* **16**(4), 229–307.
- Koppel, M. & Schler, J. (2004), Authorship verification as a one-class classification problem, in 'Proc. 21st Int. Conf. on Machine Learning', ACM Press.
- Kukushkina, O., Polikarpov, A. & Khmelev, D. (2000), 'Using literal and grammatical statistics for authorship attribution'.
- Lafferty, J. & Zhai, C. X. (2001), Document language models, query models, and risk minimization for information retrieval, in 'Proc. 24th ACM SIGIR Conf. on Research and Development in Information Retrieval', ACM Press, pp. 111–119.
- Li, J. X., Zheng, R. & Chen, H. C. (2006), 'From fingerprint to writeprint', *ACM Commun.* **49**(4), 76–82.
- Li, T., Zhu, S. H. & Ogihara, M. (2003), Efficient multi-way text categorization via generalized discriminant analysis, in 'Proc. 12th Int. Conf. on Information and Knowledge Management', ACM Press, pp. 317–324.
- Masuyama, T. & Nakagawa, H. (2004), Two step pos selection for svm based text categorization, in 'IE-ICE Transaction on Information System', Vol. E87-D.
- Pol, M. S. (2005), A stylometry-based method to measure intra and inter-authorial faithfulness for forensic applications, in 'SIGIR workshop: Stylistic Analysis of Text For Information Access'.
- Quinlan, R. (1993), *C4.5: Programs for Machine Learning*, Morgan Kaufmann.
- Sarkar, A., Roeck, A. D. & Garthwaite, P. H. (2005), Term re-occurrence measures for analyzing style, in 'SIGIR workshop: Stylistic Analysis of Text For Information Access'.
- Scholkopf, B. & Smola, A. J. (2002), *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*, MIT Press.
- Stamatatos, E., Fakotakis, N. & Kokkinakis, G. (1999), Automatic authorship attribution, in 'Proc. 9th Conf. of the European Chapter of the Association for Computational Linguistics', pp. 158–164.
- Stamatatos, E., Fakotakis, N. & Kokkinakis, G. (2001), 'Computer-based authorship attribution without lexical measures', *Computers and the Humanities* **35**(2), 193–214.
- Witten, I. H. & Frank, E. (2000), *Data Mining: Practical Machine Learning Tools with Java Implementations*, Morgan Kaufmann.
- Zhai, C. X. & Lafferty, J. (2001), A study of smoothing methods for language models applied to ad hoc information retrieval, in 'Proc. 24th ACM SIGIR Conf. on Research and Development in Information Retrieval', ACM Press, pp. 334–342.
- Zhai, C. X. & Lafferty, J. (2002), Two-stage language models for information retrieval, in 'Proc. 25th ACM SIGIR Conf. on Research and Development in Information Retrieval', ACM Press, pp. 49–56.
- Zhai, C. X. & Lafferty, J. (2004), 'A study of smoothing methods for language models applied to information retrieval', *ACM Transaction on Information System* **22**(2), 179–214.
- Zhao, Y. & Zobel, J. (2005), Effective authorship attribution using function word, in 'Proc. 2nd AIRS Asian Information Retrieval Symposium', Springer, pp. 174–190.
- Zhao, Y. & Zobel, J. (n.d.), 'Authorship search in large document collections'. Manuscript in preparation.
- Zhao, Y., Zobel, J. & Vines, P. (2006), Using relative entropy for authorship attribution, in 'Proc. 3rd AIRS Asian Information Retrieval Symposium', Springer, pp. 92–105.
- Zobel, J. & Moffat, A. (2006), 'Inverted files for text search engines', *ACM Computing Surveys*. To appear.

On Inferences of Full Hierarchical Dependencies

Sven Hartmann

Sebastian Link*

Department of Information Systems, Information Science Research Centre
Massey University, Palmerston North, New Zealand
E-mail: [S.Hartmann,S.Link]@massey.ac.nz

Abstract

Full hierarchical dependencies (FHDs) constitute a large class of relational dependencies. A relation exhibits an FHD precisely when it can be decomposed into *at least* two of its projections without loss of information. Therefore, FHDs generalise multivalued dependencies (MVDs) in which case the number of these projections is precisely two. The implication of FHDs has been defined in the context of some fixed finite universe.

This paper identifies a sound and complete set of inference rules for the implication of FHDs. This axiomatisation is very reminiscent of that for MVDs. Then, an alternative notion of FHD implication is introduced in which the underlying set of attributes is left undetermined. The main result proposes a finite axiomatisation for FHD implication in undetermined universes. Moreover, the result clarifies the role of the complementation rule as a mere means of database normalisation. In fact, an axiomatisation for FHD implication in fixed universes is proposed which allows to infer any FHDs either without using the complementation rule at all or only in the very last step of the inference. This also characterises the expressiveness of an incomplete set of inference rules in fixed universes. The results extend previous work on MVDs by Biskup.

1 Introduction

Relational databases still form the core of most database management systems, even after more than three decades following their introduction in (Codd 1970). The relational model organises data into a collection of relations. These structures permit the storage of inconsistent data, inconsistent in a semantic sense. Since this is not acceptable additional assertions, called dependencies, are formulated that every database is compelled to obey. There are many different classes of dependencies which can be utilised for improving the representation of the target database (Fagin & Vardi 1986, Thalheim 1991, Vardi 1987).

Multivalued dependencies (MVDs) (Fagin 1977, Zaniolo 1976) are an important class of dependencies. A relation exhibits an MVD precisely when it is decomposable into exactly two of its projections without loss of information (Fagin 1977). This property is fundamental to relational database design, in particular 4NF (Fagin 1977), and a lot of research

has been devoted to studying the behaviour of these dependencies. Recently, extensions of multivalued dependencies have been found very useful for various design problems in advanced data models such as the nested relational data model (Fischer, Saxton, Thomas & Van Gucht 1985), the Entity-Relationship model (Thalheim 2000), data models that support nested lists (Hartmann & Link 2004, Hartmann, Link & Schewe 2006) and XML (Vincent & Liu 2003, Vincent, Liu & Liu 2003).

Full hierarchical dependencies (FHDs) (Delobel 1978) constitute a large class of relational dependencies that subsume MVDs. A relation exhibits an FHD precisely when it is the natural join over at least two of its projections. The classical notion of an FHD (Delobel 1978) is dependent on the underlying universe R . For MVDs (Fagin 1977) their dependence on the relation schema R is syntactically reflected by the R -complementation rule which is part of the axiomatisation of MVDs (Beeri, Fagin & Howard 1977). The R -complementation rule is special in the sense that it is the only inference rule which is dependent on R . Further research on this fact has led to an alternative notion of semantic implication in which the underlying universe is left undetermined (Biskup 1980). In the same paper Biskup shows that this notion can be captured syntactically by a sound and complete set of inference rules, denoted by \mathfrak{S}_0 . If $R\mathfrak{S}_0$ results from adding the R -complementation rule to \mathfrak{S}_0 , then $R\mathfrak{S}_0$ is sound and complete for the R -implication of MVDs. In fact, every inference of an MVD by $R\mathfrak{S}_0$ can be turned into an inference of the same MVD in which the R -complementation rule is applied at most once, and if it is applied, then in the last step of the inference ($R\mathfrak{S}_0$ is said to be R -complementary). This indicates that the R -complementation rule simply reflects a part of the decomposition process, and does not necessarily infer semantically meaningful consequences.

Interestingly, research has not been continued in this direction but focused on the original notion of R -implication. Since research on dependencies seems to experience a recent revival in the context of other data models (Fischer et al. 1985, Hartmann & Link 2004, Link 2006a, Thalheim 2003, Vincent & Liu 2003, Vincent et al. 2003) it seems desirable to further extend the knowledge on relational dependencies. An advancement of such knowledge may simplify the quest of finding suitable and comprehensible extensions of relational dependencies to currently popular data models.

Contributions. In this paper we will extend the work by Biskup (Biskup 1980) from MVDs to FHDs. First, we propose a minimal complete set of sound inference rules for the implication of FHDs in fixed universes. Almost all inference rules are extensions of familiar rules from the axiomatisation of MVDs (Beeri et al. 1977). In particular, the dependence of

*This research was supported by Marsden Funding, Royal Society of New Zealand
Copyright ©2007, Australian Computer Society, Inc. This paper appeared at the Thirtieth Australasian Computer Science Conference (ACSC 2007), Ballarat, Australia. Conferences in Research and Practice in Information Technology, Vol. 62. Gillian Dobbie, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

FHDs on the underlying set R of attributes is syntactically reflected by the R -complementation rule.

Example 1.1. Suppose we design a database for a DVD collection. So far, we have decided to use attributes Title, Actor, Feature and Language. In order to model that the title of a DVD determines the set of actors independently from the rest of the information in any schema, and the title of a DVD also determines the set of DVD features independently from the rest of the information in any schema we specify the FHD

Title : $\{\{\text{Actor}\}, \{\text{Feature}\}\}$.

Note that this FHD is equivalent to the two MVDs

Title \rightarrow Actor and Title \rightarrow Feature.

If the underlying relation schema R consists of the four attributes above, then we may infer the FHDs

Title: $\{\{\text{Actor}\}, \{\text{Language}\}\}$ and
Title: $\{\{\text{Feature}\}, \{\text{Language}\}\}$

since $\{\text{Language}\}$ is the DVD-complement of $\{\text{Title}, \text{Actor}, \text{Feature}\}$. However, if we add a further attribute such as Production_Year to the schema DVD, then neither

Title: $\{\{\text{Actor}\}, \{\text{Language}\}\}$ nor
Title: $\{\{\text{Feature}\}, \{\text{Language}\}\}$

are DVD-implied by Title : $\{\{\text{Actor}\}, \{\text{Feature}\}\}$. The fundamental difference is that the latter FHD has been specified to establish a set-valued correspondence between DVD titles and the actors starring in the movie with that title (and the features available on the DVD with that title, respectively). The other two FHDs do not necessarily correspond to any semantic information, but simply result from the context in which Title, Actor, and Feature are considered. If this context is altered, then the respective FHDs disappear. \square

Example 1.1 suggests that the complementation rule is a mere means of database normalisation and does not always result in the inference of necessarily meaningful consequences. An axiomatisation for the R -implication of FHDs should therefore be complementary. The second contribution is the proposal of such a minimal axiomatisation for FHDs in fixed universes, i.e., no proper subset of inference rules is still both complete and complementary. Moreover, we investigate the impact of replacing the R -complementation rule by the so-called R -axiom. This clarifies the role of the R -complementation rule for FHDs further and extends previous work on MVDs (Biskup 1978).

One may argue that consequences that depend on the underlying universe are no consequences at all. Consequently, the notion of R -implication is not acceptable. Thus, we extend the alternative notion of implication from MVDs (Biskup 1980) to FHDs. This notion leaves the underlying set of attributes undetermined. The third contribution is the identification of a minimal complete set of sound inference rules for the implication of FHDs in undetermined universes. For instance, in Example 1.1 both FHDs

Title: $\{\{\text{Actor}\}, \{\text{Language}\}\}$ and
Title: $\{\{\text{Feature}\}, \{\text{Language}\}\}$

are DVD-implied by Title: $\{\{\text{Actor}\}, \{\text{Feature}\}\}$ where DVD = $\{\text{Title}, \text{Actor}, \text{Feature}, \text{Language}\}$, but none of the two FHDs is implied.

Previous Work. The first axiomatisation for the R -implication of MVDs was given in (Beeri

et al. 1977). The notion of implication in undetermined universes is from (Biskup 1980) in which an axiomatisation for this notion of MVD implication is given. Minimality of MVD axiomatisations are discussed in (Biskup 1978, Mendelzon 1979) in which (Biskup 1978) also introduces the R -axiom as a very weak form of the R -complementation rule. Recently, more minimal axiomatisations for both complete and complementary axiomatisations for the R -implication of MVDs, and complete axiomatisations for MVD implication in undetermined universes were studied (Hartmann & Link 2006, Link 2006a). MVDs have also been studied in the presence of the null value *no information*. In that case, (Lien 1982) was the first to propose an axiomatisation in fixed universes, and (Link 2006b) proposes a complementary axiomatisation in fixed universes, and an axiomatisation in undetermined universes. In other data models MVDs have also been investigated in the context of fixed universes. Full hierarchical dependencies were introduced in (Delobel 1978). An axiomatisation for the implication in fixed universes can be found in (Thalheim 1991, Thalheim 2000).

Organisation. The article is structured as follows. In Section 2 we will start with a brief summary of notions from the relational model of data. In particular, we repeat the notion of implication in fixed universes and summarise the axiomatisation for the implication of MVDs. Subsequently, we introduce full hierarchical dependencies as a generalisation of MVDs, and prove the completeness and minimality of a set of sound inference rules for the implication of FHDs in fixed universes. Section 3 examines the property of complementarity for FHDs. It turns out that an extension of the *subset rule* from MVDs to FHDs plays a key role in achieving complementarity. Section 4 discusses the implication of FHDs in undetermined universes. Minimality of the axiomatisations for fixed and undetermined universes are studied in Section 5. Finally, we briefly comment on possible future work in Section 6.

2 Dependencies in Fixed Universes

Let $\mathfrak{A} = \{A_1, A_2, \dots\}$ be a (countably) infinite set of symbols, called *attributes*. A *relation schema* is a finite set $R = \{A_1, \dots, A_n\}$ of distinct *attributes* from \mathfrak{A} , which represent column names of a relation. Each attribute A_i of a relation schema is associated an infinite domain $\text{dom}(A_i)$ which represents the set of possible values that can occur in the column named A_i . If X and Y are sets of attributes, then we may write XY for $X \cup Y$. If $X = \{A_1, \dots, A_m\}$, then we may write $A_1 \cdots A_m$ for X . In particular, we may write simply A to represent the singleton $\{A\}$. A *tuple* over $R = \{A_1, \dots, A_n\}$ (R -tuple or simply tuple, if R is understood) is a function $t : R \rightarrow \bigcup_{i=1}^n \text{dom}(A_i)$

with $t(A_i) \in \text{dom}(A_i)$ for $i = 1, \dots, n$. For $X \subseteq R$ let $t[X]$ denote the restriction of the tuple t over R on X , and $\text{dom}(X) = \prod_{A \in X} \text{dom}(A)$ the Cartesian product of the domains of attributes in X . A *relation* r over R is a finite set of tuples over R . The relation schema R is also called the domain $\text{Dom}(r)$ of the relation r over R . Let $r[X] = \{t[X] \mid t \in r\}$ denote the *projection* of the relation r over R on $X \subseteq R$. For $X, Y \subseteq R$, $r_1 \subseteq \text{dom}(X)$ and $r_2 \subseteq \text{dom}(Y)$ let $r_1 \bowtie r_2 = \{t \in \text{dom}(XY) \mid \exists t_1 \in r_1, t_2 \in r_2 \text{ with } t[X] = t_1[X] \text{ and } t[Y] = t_2[Y]\}$ denote the *natural join* of r_1 and r_2 . Note that the 0-ary relation $\{\emptyset\}$ is the projection $r[\emptyset]$ of r on \emptyset as well as left and right identity of the natural join operator.

Functional dependencies (FDs) between sets of

Title	Actor	Feature	Language
King Kong	Naomi Watts	Deleted Scenes	English
King Kong	Jack Black	Photo Gallery	French
King Kong	Naomi Watts	Deleted Scenes	French
King Kong	Naomi Watts	Photo Gallery	French
King Kong	Naomi Watts	Photo Gallery	English
King Kong	Jack Black	Photo Gallery	English
King Kong	Jack Black	Deleted Scene	English
King Kong	Jack Black	Deleted Scene	French

Title	Actor
King Kong	Naomi Watts
King Kong	Jack Black

Title	Feature
King Kong	Deleted Scenes
King Kong	Photo Gallery

Title	Language
King Kong	English
King Kong	French

Table 1: A DVD-relation and its projections

attributes have played a central role in the study of relational databases (Beeri & Bernstein 1979, Bernstein 1976, Bernstein & Goodman 1980, Codd 1970, Codd 1972), and seem to be central for the study of database design in other data models as well (Arenas & Libkin 2004, Hara & Davidson 1999, Levene & Loizou 1998, Link 2006a, Tari, Stokes & Spaccapietra 1997, Weddell 1992, Wijzen 1999). The notion of a functional dependency is well-understood and the semantic interaction between these dependencies has been syntactically captured by Armstrong's well-known axioms (Armstrong 1974, Armstrong, Nakamura & Rudnicki 2002). A *functional dependency* (FD) (Codd 1972) on the relation schema R is an expression $X \rightarrow Y$ where $X, Y \subseteq R$. A relation r over R satisfies the FD $X \rightarrow Y$, denoted by $\models_r X \rightarrow Y$, if and only if every pair of tuples in r that agrees on each of the attributes in X also agrees on the attributes in Y . That is, $\models_r X \rightarrow Y$ if and only if $t_1[Y] = t_2[Y]$ whenever $t_1[X] = t_2[X]$ holds for any $t_1, t_2 \in r$.

FDs are incapable of modelling many important properties that database users have in mind. Multivalued dependencies (MVDs) provide a more general notion and offer a response to the shortcomings of FDs. A *multivalued dependency* (MVD) (Fagin 1977, Zaniolo 1976) on R is an expression $X \twoheadrightarrow Y$ where $X, Y \subseteq R$. A relation r over R satisfies the MVD $X \twoheadrightarrow Y$, denoted by $\models_r X \twoheadrightarrow Y$, if and only if for all $t_1, t_2 \in r$ with $t_1[X] = t_2[X]$ there is some $t \in r$ with $t[XY] = t_1[XY]$ and $t[X(R - Y)] = t_2[X(R - Y)]$. Informally, the relation r satisfies $X \twoheadrightarrow Y$ when the value on X determines the set of values on Y independently from the set of values on $R - Y$. This actually suggests that the relation schema R is overloaded in the sense that it carries two independent facts XY and $X(R - Y)$. More precisely, it is shown in (Fagin 1977) that MVDs “provide a necessary and sufficient condition for a relation to be decomposable into two of its projections without loss of information (in the sense that the original relation is guaranteed to be the join of the two projections)”. This means that $\models_r X \twoheadrightarrow Y$ if and only if $r = r[XY] \bowtie r[X(R - Y)]$. This characteristic of MVDs is fundamental to relational database design and 4NF (Fagin 1977). A lot of research has therefore been devoted to studying the behaviour of these dependencies.

Full hierarchical dependencies generalise multivalued dependencies (Delobel 1978).

Definition 2.1. A full hierarchical dependency on the relation schema R is an expression $X : S$ where $X \subseteq R$ and S is a non-empty set of pairwise disjoint subsets of R that are also disjoint from X , i.e., $S \neq \emptyset$, for all $Y \in S$ we have $Y \subseteq R$ and for all $Y, Z \in S \cup \{X\}$ we have $Y \cap Z = \emptyset$. An R -relation $r \subseteq \text{dom}(R)$ is said to satisfy (or said to be a model of) the full hierarchical dependency $X : \{Y_1, \dots, Y_k\}$ on

R , denoted by $\models_r X : \{Y_1, \dots, Y_k\}$, if and only if for all $t_1, \dots, t_{k+1} \in r$ the following condition is satisfied: if $t_i[X] = t_j[X]$ for all $1 \leq i, j \leq k+1$, then there is some $t \in r$ such that $t[XY_i] = t_i[XY_i]$ for $i = 1, \dots, k$ and $t[X(R - XY_1 \dots Y_k)] = t_{k+1}[X(R - XY_1 \dots Y_k)]$. \square

Notice that Definition 2.1 reduces to the definition of MVDs in case that $k = 1$. Note that our definition of FHDs is slightly different from the original definition (Delobel 1978). There, an FHD is defined as an expression

$$X : Y_1 \mid \dots \mid Y_k$$

such that X, Y_1, \dots, Y_k form a partition of the underlying relation schema R . Our definition is different in two aspects. Firstly, $X, Y_1, \dots, Y_k, R - XY_1 \dots Y_k$ form a partition of R in our definition. Secondly, the right-handed side of an FHD is a set system $\{Y_1, \dots, Y_k\}$ over R in our definition, i.e., there is no sequence of attribute sets Y_1, \dots, Y_k as in the original definition (Delobel 1978). These two differences result in a simpler axiomatisation and the correspondence to the original definition of MVDs is much stronger (Fagin 1977). The next theorem illustrates the importance of FHDs for the removal of redundant data.

Theorem 2.1. Let $X, Y_1, \dots, Y_k \subseteq R$ be pairwise disjoint and $k \geq 1$. An R -relation r satisfies the FHD $X : \{Y_1, \dots, Y_k\}$ on R if and only if $r = r[XY_1] \bowtie \dots \bowtie r[XY_k] \bowtie r[X(R - XY_1 \dots Y_k)]$. \square

Example 2.1. Consider again the relation schema DVD comprising the four attributes Title, Actor, Feature and Language. Table 1 shows a relation r that satisfies the FHD

$$\text{Title} : \{\{\text{Actor}\}, \{\text{Feature}\}\}$$

and the projections of r on $\{\text{Title}, \text{Actor}\}$, $\{\text{Title}, \text{Feature}\}$ and $\{\text{Title}, \text{Language}\}$, respectively. Indeed, r is the natural join of its projections on these three attribute sets. \square

Theorem 2.2. Let $X, Y_1, \dots, Y_k \subseteq R$ be pairwise disjoint and $k \geq 1$. An R -relation r satisfies the FHD $X : \{Y_1, \dots, Y_k\}$ on R if and only if for all $i = 1, \dots, k$, r satisfies the MVD $X \twoheadrightarrow Y_i$ on R . \square

Example 2.2. Consider again the relation r in Table 1. In particular, one can verify that r satisfies the MVDs $\text{Title} \twoheadrightarrow \text{Actor}$ and $\text{Title} \twoheadrightarrow \text{Feature}$. \square

For the design of a relational database schema dependencies are normally specified as semantic constraints on the relations which are intended to be instances of the schema. During the design process one usually needs to determine further dependencies which are logically implied by the given ones. In order to emphasise the dependence of implication from the underlying relation schema R we refer to R -implication.

Definition 2.2. Let $\Sigma = \{X_1 : \{Y_1^1, \dots, Y_{l_1}^1\}, \dots, X_n : \{Y_1^n, \dots, Y_{l_n}^n\}\}$ and $X : \{Y_1, \dots, Y_k\}$ be FHDs on the relation schema R , i.e.,

$$X \cup \bigcup_{i=1}^k Y_i \cup \bigcup_{j=1}^n \left(X_j \cup \bigcup_{s=1}^{l_j} Y_s^j \right) \subseteq R.$$

Then Σ R -implies $X : \{Y_1, \dots, Y_k\}$ if and only if each relation r over R that satisfies all FHDs in Σ also satisfies $X : \{Y_1, \dots, Y_k\}$. \square

Notice that Definition 2.2 covers MVDs in case that $l_j = 1$ for $j = 1, \dots, n$ and $k = 1$ (Biskup 1980). In order to determine all logical consequences of a set of MVDs one can use the following set of inference rules for the R -implication of multivalued dependencies (Beeri et al. 1977). These *inference rules* have the form

$$\frac{\text{premise}}{\text{conclusion}}$$

and inference rules without a premise are called *axioms*. Note that we use the natural complementation rule (Biskup 1978) instead of the complementation rule that was originally proposed (Beeri et al. 1977).

$$\begin{array}{c} \frac{}{X \rightarrow Y}^{Y \subseteq X} \quad \frac{X \rightarrow Y}{XU \rightarrow YV}^{V \subseteq U} \\ \text{(reflexivity, } \mathcal{R}_{\text{MVD}}) \quad \text{(augmentation, } \mathcal{A}_{\text{MVD}}) \\ \\ \frac{X \rightarrow Y, Y \rightarrow Z}{X \rightarrow Z - Y} \quad \frac{X \rightarrow Y, W \rightarrow Z}{X \rightarrow Z - Y}^{Y \cap W = \emptyset} \\ \text{(pseudo-transitivity, } \mathcal{T}_{\text{MVD}}) \quad \text{(subset, } \mathcal{S}_{\text{MVD}}) \\ \\ \frac{X \rightarrow Y}{X \rightarrow R - Y} \quad \frac{X \rightarrow Y, X \rightarrow Z}{X \rightarrow YZ} \\ \text{(R-complementation, } \mathcal{C}_R^{\text{MVD}}) \quad \text{(union, } \mathcal{U}_{\text{MVD}}) \\ \\ \frac{X \rightarrow Y, X \rightarrow Z}{X \rightarrow Z - Y} \quad \frac{X \rightarrow Y, X \rightarrow Z}{X \rightarrow Y \cap Z} \\ \text{(difference, } \mathcal{D}_{\text{MVD}}) \quad \text{(intersection, } \mathcal{I}_{\text{MVD}}) \end{array}$$

The set $\{\mathcal{R}_{\text{MVD}}, \mathcal{A}_{\text{MVD}}, \mathcal{T}_{\text{MVD}}, \mathcal{C}_R^{\text{MVD}}, \mathcal{U}_{\text{MVD}}, \mathcal{D}_{\text{MVD}}, \mathcal{I}_{\text{MVD}}\}$ is both sound and complete for the R -implication of MVDs (Beeri et al. 1977). Let $\Sigma \cup \{\sigma\}$ be a set of dependencies from the class \mathcal{C} on the relation schema R . Let $\Sigma \vdash_{\mathfrak{S}} \sigma$ denote the inference of σ from a set Σ of dependencies from \mathcal{C} with respect to the set \mathfrak{S} of inference rules. Let $\Sigma_{\mathfrak{S}}^+ = \{\sigma \mid \Sigma \vdash_{\mathfrak{S}} \sigma\}$ denote the *syntactic hull* of Σ under inference using only rules from \mathfrak{S} . In what follows we use the letter R to emphasise the fact that a set $R\mathfrak{S}$ refers to R -implication, i.e., to implication in fixed universes. An inference rule is called *R -sound* if the set of dependencies in the premise of the rule R -implies the dependency in the conclusion. It is well-known that all the rules above are R -sound for all R (i.e. if one restricts U, V, W, X, Y, Z to be subsets of R) (Beeri et al. 1977). The set $R\mathfrak{S}$ is called *R -sound* for the R -implication of dependencies from \mathcal{C} if and only if for every set Σ of dependencies from \mathcal{C} on the relation schema R we have $\Sigma_{R\mathfrak{S}}^+ \subseteq \Sigma_R^* = \{\sigma \in \mathcal{C} \mid \Sigma R\text{-implies } \sigma\}$. The set $R\mathfrak{S}$ is called *R -complete* for the R -implication of dependencies from \mathcal{C} if and only if for every set Σ of dependencies from \mathcal{C} on R we have $\Sigma_R^* \subseteq \Sigma_{R\mathfrak{S}}^+$. Furthermore, the set $R\mathfrak{S}$ is called *complete (sound)* for the R -implication of dependencies from \mathcal{C} if and only if it is R -complete (R -sound) for the R -implication of dependencies from \mathcal{C} for all relation schemata R .

An interesting question is now whether all the rules of a certain set of inference rules are really necessary to capture the R -implication of dependencies from \mathcal{C} for every relation schema R . More precisely, an inference rule \mathfrak{R} is said to be *R -independent* from the set $R\mathfrak{S}$ if and only if there is some set $\Sigma \cup \{\sigma\}$ of dependencies from \mathcal{C} on the relation schema R such that $\sigma \notin \Sigma_{R\mathfrak{S}}^+$, but $\sigma \in \Sigma_{R\mathfrak{S} \cup \{\mathfrak{R}\}}^+$. Moreover, an inference rule \mathfrak{R} is said to be *independent* from $R\mathfrak{S}$ if and only if there is some relation schema R such that \mathfrak{R} is R -independent from $R\mathfrak{S}$. A complete set $R\mathfrak{S}$ is called *minimal* for the R -implication of dependencies from \mathcal{C} if and only if every inference rule $\mathfrak{R} \in R\mathfrak{S}$ is independent from $R\mathfrak{S} - \{\mathfrak{R}\}$. This means that no proper subset of $R\mathfrak{S}$ is still complete.

2.1 Sound Inference Rules for FHDs in fixed Universes

We will denote the set of inference rules from Table 2 by $R\mathfrak{S}$.

Theorem 2.3. The set $R\mathfrak{S}$ is sound for the R -implication of FHDs.

Sketch. We need to show that for an arbitrary relation schema R , and an arbitrary set Σ of FHDs on R we have $\Sigma_{R\mathfrak{S}}^+ \subseteq \Sigma_R^*$. The proof will make extensive use of Theorem 2.2 and the R -soundness of the inference rules for MVDs (Beeri et al. 1977). Notice that it is sufficient to show the R -soundness of each inference rule. Subsequently, one can show by induction that each FHD occurring in an inference by $R\mathfrak{S}$ is also R -implied by Σ .

We only show how to prove the *augmentation rule* \mathcal{A} . Let r be some arbitrary R -relation that satisfies $X : \{Y_1, \dots, Y_k\}$. Theorem 2.2 tells us that r also satisfies $X \rightarrow Y_i$ for all $i = 1, \dots, k$. The soundness of the augmentation rule \mathcal{A}_{MVD} shows that r also satisfies $XZ \rightarrow Y_i$ for all $i = 1, \dots, k$, and the soundness of the reflexivity axiom \mathcal{R}_{MVD} identifies r as a model of $XZ \rightarrow Z$. We conclude by the soundness of the difference rule \mathcal{D}_{MVD} that r satisfies $XZ \rightarrow Y_i - Z$ for all $i = 1, \dots, k$. Consequently, $\models_r XZ : \{Y_1 - Z, \dots, Y_k - Z\}$ by Theorem 2.2.

The proof for the remaining rules is similar. \square

Lemma 2.1. The following inference rules are derivable from $R\mathfrak{S}$

$$\frac{X : \{Y_1, \dots, Y_k\}}{X : \{Y_1, \dots, Y_k, \emptyset\}} \quad \frac{X : \{Y_1, \dots, Y_k, Y_{k+1}\}}{X : \{Y_1, \dots, Y_k Y_{k+1}\}} \\ \text{(empty-set-introduction, } \mathcal{I}_{\emptyset}) \quad \text{(merging, } \mathcal{M})$$

and thus are sound for the R -implication of FHDs.

Proof. The *empty-set-introduction rule* \mathcal{I}_{\emptyset} is derivable from $R_{\emptyset}, \mathcal{A}$ and \mathcal{T} .

$$\frac{\frac{\emptyset : \{\emptyset\}}{X : \{\emptyset\}}^{\mathcal{R}_{\emptyset}} \quad \frac{X : \{Y_1, \dots, Y_k\}}{X \cup \emptyset : \{Y_1, \dots, Y_k\}}^{\mathcal{A}}}{X : \{Y_1, \dots, Y_k, \emptyset\}}^{\mathcal{T}}$$

The *merging rule* \mathcal{M} is derivable from \mathcal{O} and \mathcal{U} .

$$\frac{\frac{X : \{Y_1, \dots, Y_k, Y_{k+1}\}}{X : \{Y_1, \dots, Y_k\}}^{\mathcal{O}} \quad \frac{X : \{Y_1, \dots, Y_k, Y_{k+1}\}}{X : \{Y_{k+1}\}}^{\mathcal{O}}}{X : \{Y_1, \dots, Y_{k-1}, Y_k Y_{k+1}\}}^{\mathcal{U}}$$

This concludes the proof. \square

$\frac{}{\emptyset : \{\emptyset\}}$ (empty-set-axiom, \mathcal{R}_\emptyset)	$\frac{X : \{Y_1, \dots, Y_k\}}{XZ : \{Y_1 - Z, \dots, Y_k - Z\}}$ (augmentation, \mathcal{A})	$\frac{XY : \{Y_1, \dots, Y_k\}, X : \{Y\}}{X : \{Y_1, \dots, Y_k, Y\}}$ (transitivity, \mathcal{T})
$\frac{X : \{Y_1, \dots, Y_k\}}{X : \{Y_1, \dots, Y_{k-1}, R - XY_1 \dots Y_k\}}$ (R -complementation, \mathcal{C}_R)	$\frac{X : \{Y_1, \dots, Y_k, Y\}}{X : \{Y_1, \dots, Y_k\}}$ (omission, \mathcal{O})	
$\frac{X : \{Y_1, \dots, Y_k\}, X : \{Z\}}{X : \{Y_1 - Z, \dots, Y_{k-1} - Z, Y_k Z\}}$ (union, \mathcal{U})	$\frac{X : \{Y_1, \dots, Y_k\}, X : \{Z\}}{X : \{Y_1, \dots, Y_{k-1}, Y_k - Z\}}$ (difference, \mathcal{D})	$\frac{X : \{Y_1, \dots, Y_k\}, X : \{Z\}}{X : \{Y_1, \dots, Y_{k-1}, Y_k \cap Z\}}$ (intersection, \mathcal{I})

Table 2: Inference rules for the R -implication of FHDs

2.2 Completeness

Let R be some arbitrary relation schema, and let Σ be a set of FHDs on R . Let $Dep_R(X)$ be the set of all $W \subseteq R$ for which some FHD $X : S$ with $W \in S$ is derivable from Σ using the inference rules $R\mathfrak{S}$, i.e., $Dep_R(X) = \{W \subseteq R \mid X : S \in \Sigma_{R\mathfrak{S}}^+ \text{ and } W \in S\}$. Note that $Dep_R(X)$ is finite, and $(Dep_R(X), \subseteq, \cup, \cap, (\cdot)^c, \emptyset, R)$ constitutes a Boolean algebra due to the soundness of union, difference and intersection rule. Recall that an element $a \in P$ of a poset $(P, \subseteq, 0)$ with least element 0 is called an *atom* of $(P, \subseteq, 0)$ (Graetzer 1998) if and only if $a \neq 0$ and every element $b \in P$ with $b \subseteq a$ satisfies $b = 0$ or $b = a$. $(P, \subseteq, 0)$ is called *atomic* if and only if for every element $b \in P$ with $b \neq 0$ there is an atom $a \in P$ with $a \subseteq b$. In particular, every finite Boolean algebra is atomic. The set $Dep_{B_R}(X)$ of all atoms of $(Dep_R(X), \subseteq, \emptyset)$ is called the *dependency basis* of X with respect to Σ (Beeri 1980).

Theorem 2.4. *Let $\Sigma \cup \{X : S\}$ be a set of FHDs on the relation schema R . Then $X : S \in \Sigma_{R\mathfrak{S}}^+$ if and only if for every $Y \in S$ there is some $\mathcal{Y} \subseteq Dep_{B_R}(X)$ such that $Y = \bigcup \mathcal{Y}$.*

Proof. Let $Y \in S$ for $X : S \in \Sigma_{R\mathfrak{S}}^+$. That is, $Y \in Dep_R(X)$, and since every element b of a Boolean algebra is the join over those atoms a with $a \leq b$ we know that $Y = \bigcup \mathcal{Y}$ for $\mathcal{Y} = \{W \in Dep_{B_R}(X) \mid W \subseteq Y\}$.

Vice versa, let $Y = \bigcup \mathcal{Y}$ for some $\mathcal{Y} \subseteq Dep_{B_R}(X)$. Since $Dep_{B_R}(X) \subseteq Dep_R(X)$ and $Dep_R(X)$ is closed under unions it follows that $Y \in Dep_R(X)$. As this is true for all $Y \in S$ and the sets in S are pairwise disjoint we can apply the *empty-set-introduction rule* \mathcal{R}_\emptyset and the *union rule* \mathcal{U} to infer $X : S \in \Sigma_{R\mathfrak{S}}^+$. \square

Theorem 2.5. *The set $R\mathfrak{S}$ of inference rules is complete for the R -implication of FHDs.* \square

2.3 A minimal Axiomatisation

The objective is to identify a minimal subset of $R\mathfrak{S}$, i.e., a set in which each single rule is essential and not derivable from the rest of the rules.

Theorem 2.6. *The set $\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{C}_R\}$ consisting of the empty-set axiom, the augmentation rule, the transitivity rule, the omission rule and the R -complementation rule is minimal for the R -implication of FHDs.*

We prove Theorem 2.6 by a series of lemmata. First, it is shown that union rule \mathcal{U} , intersection rule \mathcal{I} and difference rule \mathcal{D} are derivable from $\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{C}_R\}$. Subsequently, it is demonstrated that none of the rules in $\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{C}_R\}$ can

be omitted without losing completeness for the R -implication of FHDs.

Lemma 2.2. *The union rule \mathcal{U} is derivable from $\{\mathcal{A}, \mathcal{T}, \mathcal{C}_R, \mathcal{O}\}$.* \square

Lemma 2.3. *The intersection rule \mathcal{I} is derivable from $\{\mathcal{A}, \mathcal{T}, \mathcal{C}_R, \mathcal{O}, \mathcal{U}\}$.* \square

Lemma 2.4. *The difference rule \mathcal{D} is derivable from $\{\mathcal{C}_R, \mathcal{I}\}$.*

Proof. Note that $Y_k \cap (R - XZ) = Y_k - XZ = Y_k - Z$ since $Y_k \cap X = \emptyset$.

$$\frac{X : \{Y_1, \dots, Y_k\} \quad \frac{X : \{Z\}}{X : \{R - XZ\}}^{\mathcal{C}_R}}{X : \{Y_1, \dots, Y_{k-1}, Y_k - Z\}}^{\mathcal{I}}$$

This concludes the proof. \square

Theorem 2.5 and Lemmata 2.2, 2.3 and 2.4 show that $\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{C}_R\}$ is indeed complete for the R -implication of FHDs. In order to complete the proof of Theorem 2.6 it remains to verify the independence of every inference rule in $\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{C}_R\}$ from the rest of the rules.

Lemma 2.5. *The empty-set-axiom \mathcal{R}_\emptyset is independent from $\{\mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{C}_R\}$.*

Proof. Let $R = \emptyset$, $\Sigma = \emptyset$ and $\sigma = \emptyset : \{\emptyset\}$. It follows that $\sigma \notin \Sigma_{\{\mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{C}_R\}}^+$, but $\sigma \in \Sigma_{\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{C}_R\}}^+$. \square

Lemma 2.6. *The augmentation rule \mathcal{A} is independent from $\{\mathcal{R}_\emptyset, \mathcal{T}, \mathcal{O}, \mathcal{C}_R\}$.*

Proof. Let $R = A$, $\Sigma = \emptyset$ and $\sigma = A : \{\emptyset\}$. The closure $\Sigma_{\mathfrak{S}}^+$ of a set \mathfrak{S} of sound inference rules is represented as a table. The FHD $X : \{Y_1, \dots, Y_k\}$ belongs to $\Sigma_{\mathfrak{S}}^+$ if and only if the entry in row labelled X and column labelled $\{Y_1, \dots, Y_k\}$ is a cross \times . Notice that there are some cells which do not represent any FHD. These have the entry \star . The closure $\Sigma_{\{\mathcal{R}_\emptyset, \mathcal{T}, \mathcal{O}, \mathcal{C}_R\}}^+$ can be obtained as follows. The *empty-set-axiom* \mathcal{R}_\emptyset yields the FHD $\emptyset : \{\emptyset\}$. A subsequent application of the *R -complementation rule* \mathcal{C}_R allows us to obtain the FHD $\emptyset : \{A\}$. Finally, the *transitivity rule* \mathcal{T} can be applied to $\emptyset : \{\emptyset\}$ and $\emptyset : \{A\}$ to infer $\emptyset : \{A, \emptyset\}$. This set is closed under derivation using $\{\mathcal{R}_\emptyset, \mathcal{T}, \mathcal{O}, \mathcal{C}_R\}$.

	$\{\emptyset\}$	$\{A\}$	$\{A, \emptyset\}$
\emptyset	\times	\times	\times
A		\star	\star

It follows that $\sigma \notin \Sigma_{\{\mathcal{R}_\emptyset, \mathcal{T}, \mathcal{O}, \mathcal{C}_R\}}^+$ but $\sigma \in \Sigma_{\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{C}_R\}}^+$. \square

Lemma 2.7. *The transitivity rule \mathcal{T} is independent from $\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{O}, \mathcal{C}_R\}$.*

Proof. Let $R = A$, $\Sigma = \emptyset$ and $\sigma = \emptyset : \{A, \emptyset\}$. The closure $\Sigma_{\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{O}, \mathcal{C}_R\}}^+$ can be obtained as follows. The *empty-set-axiom* \mathcal{R}_\emptyset yields the FHD $\emptyset : \{\emptyset\}$. A subsequent application of the *R-complementation rule* \mathcal{C}_R allows us to obtain the FHD $\emptyset : \{A\}$. Finally, the *augmentation rule* \mathcal{A} can be applied to $\emptyset : \{\emptyset\}$ to infer $A : \{\emptyset\}$. This set is closed under derivation using $\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{O}, \mathcal{C}_R\}$.

	$\{\emptyset\}$	$\{A\}$	$\{A, \emptyset\}$
\emptyset	×	×	
A	×	★	★

It follows that $\sigma \notin \Sigma_{\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{O}, \mathcal{C}_R\}}^+$ but $\sigma \in \Sigma_{\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{C}_R\}}^+$. \square

Lemma 2.8. *The omission rule \mathcal{O} is independent from $\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{C}_R\}$.*

Proof. In this case we can choose $R = AB$, $\Sigma = \{\emptyset : \{\emptyset, A\}\}$ and $\sigma = \emptyset : \{A\}$. \square

Lemma 2.9. *The R-complementation rule \mathcal{C}_R is independent from $\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}\}$.*

Proof. Let $R = A$, $\Sigma = \emptyset$ and $\sigma = \emptyset : \{A\}$. The closure $\Sigma_{\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}\}}^+$ can be obtained as follows. The *empty-set-axiom* \mathcal{R}_\emptyset yields the FHD $\emptyset : \{\emptyset\}$. A subsequent application of the *augmentation rule* \mathcal{A} allows us to derive the FHD $A : \{\emptyset\}$. This set is closed under derivation using $\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}\}$.

	$\{\emptyset\}$	$\{A\}$	$\{A, \emptyset\}$
\emptyset	×		
A	×	★	★

It follows that $\sigma \notin \Sigma_{\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}\}}^+$ but $\sigma \in \Sigma_{\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{C}_R\}}^+$. \square

2.4 A weaker version of R-complementation

Biskup (Biskup 1978) has replaced the *R-complementation rule* \mathcal{C}_R by the so-called *R-axiom* $\frac{}{\emptyset \rightarrow R}$ and still obtained a complete axiomatisation for the *R-implication* of MVDs. That is, *R-axiom*, *augmentation rule* and *pseudo-transitivity rule* form a sound and complete set of inference rules for the *R-implication* of MVDs (Biskup 1978).

Let $\frac{}{\emptyset : \{R\}}$ be the *R-axiom* for FHDs. This inference rule is sound for the *R-implication* of FHDs. As it turns out we can simply replace the *R-complementation rule* \mathcal{C}_R by the *R-axiom* and still maintain completeness.

Theorem 2.7. *The set $\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}, \text{R-axiom}\}$ consisting of the empty-set axiom, the augmentation rule, the transitivity rule, the omission rule and the R-axiom is sound and complete for the R-implication of FHDs.*

Proof. One can show that the *R-complementation rule* \mathcal{C}_R is derivable from the *R-axiom*, the *augmentation rule*, the *transitivity rule*, and the *omission rule*. The statement is then a consequence of Theorem 2.5. \square

3 A complementary Axiomatisation

We have seen before that the set $\{\mathcal{R}, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{C}_R\}$ is a minimal axiomatisation for the *R-implication* of FHDs. However, Example 1.1 has brought up a possible difficulty with this system. If the *R-complementation rule* simply represents the database normalisation process, then this should be reflected by any axiomatisation. More precisely, an application of the *R-complementation rule* \mathcal{C}_R during any inferences should be restricted to the very last step of this inference (if needed at all). This would ensure that no possibly semantically meaningless information could be derived. A complete set $R\mathfrak{S}$ of inference rules is said to be *complementary* for the *R-implication* of FHDs if and only if it is *R-complementary* for every relation schema R . That is, for every set $\Sigma \cup \{\sigma\}$ of FHDs on R the inference of σ from Σ using $R\mathfrak{S}$ can be turned into an inference of σ from Σ using $R\mathfrak{S}$ in which the *R-complementation rule* \mathcal{C}_R is applied at most once, and if it is applied, then it is applied in the last step of the inference. As it turns out the system $\{\mathcal{R}, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{C}_R\}$ is not complementary.

Example 3.1. *Let Σ consist of the two FHDs*

Title : {Actor, Feature} and Title : {Actor, Language}.

It can be shown that

$$\text{Title : \{Actor, Feature Language\}} \notin \Sigma_{\{\mathcal{R}, \mathcal{A}, \mathcal{T}, \mathcal{O}\}}^+,$$

see Lemma 5.7, and

$$\text{Title : \{Actor, Y\}} \notin \Sigma_{\{\mathcal{R}, \mathcal{A}, \mathcal{T}, \mathcal{O}\}}^+$$

for any Y such that

$$Y - \{\text{Title, Actor, Feature, Language}\} \neq \emptyset,$$

see Lemma 4.1.

For DVD={Title, Actor, Feature, Language, Crew} we have

$$\text{Title : \{Actor, Feature Language\}} \in \Sigma_{\{\mathcal{R}, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{C}_R\}}^+.$$

Hence, in any such inference the DVD-complementation rule \mathcal{C}_{DVD} must be applied at least once. However, since

$$\text{DVD - \{Title, Actor, Feature, Language\} = \{Crew\}}$$

\mathcal{C}_{DVD} is not just used as the last rule. \square

The goal is to find an axiomatisation for the *R-implication* of FHDs that soundly reflects the role of the *R-complementation rule* \mathcal{C}_R as a mere means of database normalisation. That is, we would like to identify a complementary axiomatisation for FHDs. This objective has been successfully achieved in the case of MVDs (Biskup 1980, Link 2006b). A key role played the so-called *subset rule* \mathcal{S}_{MVD} which we now generalise into the framework of FHDs.

$$\frac{X : Y, W : \{Y_1, \dots, Y_k\}}{X : \{Y_1 \cap Y, \dots, Y_k \cap Y, Y - Y_1 \dots Y_k\}} Y \cap W = \emptyset$$

(subset, \mathcal{S})

It is not obvious why the set $Y - Y_1 \dots Y_k$ is included in the conclusion of the *subset rule*. In fact, this set is needed in the proof of Theorem 3.1 to shift applications of the *R-complementation rule* to the end of an inference.

Lemma 3.1. *The subset rule \mathcal{S} is sound for the R-implication of FHDs.*

Proof. Let r be some arbitrary relation such that $X \cup Y \cup W \cup \bigcup_{i=1}^k Y_i \subseteq R$. Let r satisfy $X : Y$ and $W : \{Y_1, \dots, Y_k\}$ where $Y \cap W = \emptyset$ holds. We know by Theorem 2.2 that r satisfies $X \rightarrow Y$ and $W \rightarrow Y_i$ for all $i = 1, \dots, k$. We conclude by the soundness of the subset rule \mathcal{S}_{MVD} that $\models_r X \rightarrow Y_i \cap Y$ for all $i = 1, \dots, k$. Moreover, the soundness of the difference rule \mathcal{D}_{MVD} shows that r satisfies $X \rightarrow Y - Y_i$ for all $i = 1, \dots, k$ since $Y - (Y_i \cap Y) = Y - Y_i$ holds. A further application of Theorem 2.2 shows that r satisfies $X : \{Y_1 \cap Y, \dots, Y_k \cap Y, Y - Y_1 \dots Y_k\}$. \square

Let Σ be a set of FHDs, and let \mathfrak{S} be a set of inference rules. A finite sequence of FHDs $\gamma = [\sigma_1, \dots, \sigma_n]$ is called an *inference from Σ by \mathfrak{S}* if and only if each σ_i is either an element of Σ or is obtained by applying one of the rules of \mathfrak{S} to appropriate elements of $\{\sigma_1, \dots, \sigma_{i-1}\}$. We say that the inference γ infers σ_n , i.e., the last element of the sequence γ . In fact, $\Sigma_{\mathfrak{S}}^+$ denotes the set of all FHDs which are inferred by some inference from Σ by \mathfrak{S} .

Theorem 3.1. *Let Σ be a set of FHDs. For each inference γ from Σ by the system $R\mathfrak{H} = \{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{C}_R\}$ there is an inference ξ from Σ by the system $R\mathfrak{H}_C = \{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{S}, \mathcal{M}, \mathcal{C}_R\}$ with the following properties:*

- γ and ξ infer the same FHD,
- in ξ the R -complementation rule \mathcal{C}_R is applied at most once, and
- if \mathcal{C}_R is applied in ξ , then \mathcal{C}_R is applied as the last rule.

Proof. We proceed by induction on the length l of γ . If $l = 1$, then $\xi := \gamma$ has the desired properties. Let $l > 1$, and $\gamma = [\sigma_1, \dots, \sigma_l]$ be an inference from Σ by $R\mathfrak{H}$ which has length l . One would need to consider five cases according to which inference rule in $R\mathfrak{H}$ was applied to infer σ_l from $[\sigma_1, \dots, \sigma_{l-1}]$. However, due to lack of space we only consider the case where we infer σ_l by applying the *omission rule* \mathcal{O} to the premise σ_i with $i < l$. Let ξ_i be obtained by using the induction hypothesis for $\gamma_i := [\sigma_1, \dots, \sigma_i]$.

Consider the inference $\xi := [\xi_i, \sigma_l]$. If \mathcal{C}_R is not applied in ξ_i , then ξ has the desired properties. If \mathcal{C}_R is applied in ξ_i (as the last rule), then the last two steps of ξ either have the form:

$$\frac{\frac{X : \{Y_1, \dots, Y_k, Y\}}{X : \{Y_1, \dots, Y_k, R - XY Y_1 \dots Y_k\}} \mathcal{C}_R}{X : \{Y_1, \dots, Y_k\}} \mathcal{O}.$$

or

$$\frac{\frac{X : \{Y_1, \dots, Y_k, Y\}}{X : \{Y_1, \dots, Y_k, R - XY Y_1 \dots Y_k\}} \mathcal{C}_R}{XZ : \{Y_1, \dots, Y_{i-1}, Y_{i+1}, \dots, Y_k, R - XY Y_1 \dots Y_k\}} \mathcal{O}.$$

In the first case these steps may be simply replaced by

$$\frac{X : \{Y_1, \dots, Y_k, Y\}}{X : \{Y_1, \dots, Y_k\}} \mathcal{O}.$$

In the second case, these steps can be replaced as follows:

$$\frac{\frac{X : \{Y_1, \dots, Y_k, Y\}}{X : \{Y_1, \dots, Y_{i-1}, Y_{i+1}, \dots, Y_k, Y_i Y\}} \mathcal{M}}{X : \{Y_1, \dots, Y_{i-1}, Y_{i+1}, \dots, Y_k, R - XY Y_1 \dots Y_k\}} \mathcal{C}_R.$$

In both cases the result of these replacements is an inference with the desired properties. \square

Corollary 3.1. *The axiomatisation $R\mathfrak{H}_C = \{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{S}, \mathcal{M}, \mathcal{C}_R\}$ is complementary for the R -implication of FHDs.* \square

Theorem 3.1 states that $R\mathfrak{H}_C$ is almost complete for the R -implication of FHDs.

Corollary 3.2. *Let $R \subseteq \mathfrak{A}$ be a finite set of attributes. Then for all finite sets $\Sigma = \{X_1 : \{Y_1^1, \dots, Y_{l_1}^1\}, \dots, X_n : \{Y_1^n, \dots, Y_{l_n}^n\}\}$ of FHDs, for all FHDs $X : \{Y_1, \dots, Y_k\}$ such that $X \cup \bigcup_{i=1}^k Y_i \cup$*

$\bigcup_{j=1}^n \left(X_j \cup \bigcup_{s=1}^{l_j} Y_s^j \right) \subseteq R$ we have: $X : \{Y_1, \dots, Y_k\} \in \Sigma_{R\mathfrak{H}_C}^+$ if and only if $X : \{Y_1, \dots, Y_k\} \in \Sigma_{\mathfrak{H}_C}^+$ or there is some i such that $1 \leq i \leq k$ and $X : \{Y_1, \dots, Y_{i-1}, Y_{i+1}, \dots, Y_k, R - XY_1 \dots Y_k\} \in \Sigma_{\mathfrak{H}_C}^+$. \square

4 FHDs in undetermined universes

We will now investigate the alternative notion of implication in which the underlying set of attributes is undetermined. Notice that this form of implication has been studied for MVDs (Biskup 1980, Link 2006a).

According to Example 1.1 it may be argued that consequences which are dependent on the underlying relation schema are in fact no consequences at all. This implies, however, that the notion of R -implication is not suitable. This observation already applies to MVDs (Biskup 1980, Link 2006a), and Biskup introduced an alternative notion of MVD implication (Biskup 1980) in which the underlying set of attributes remains undetermined. We will now extend this notion to full hierarchical dependencies.

An FHD is a syntactic expression $X : \{Y_1, \dots, Y_k\}$ with $X, Y_1, \dots, Y_k \subseteq \mathfrak{A}$. The FHD $X : \{Y_1, \dots, Y_k\}$ is satisfied by some relation r if and only if $X \cup \bigcup_{i=1}^k Y_i \subseteq \text{Dom}(r)$ and

$$r = r[X Y_1] \bowtie \dots \bowtie r[X Y_k] \bowtie r[X \cup (\text{Dom}(r) - X Y_1 \dots Y_k)].$$

Definition 4.1. *The set $\Sigma = \{X_1 : \{Y_1^1, \dots, Y_{l_1}^1\}, \dots, X_n : \{Y_1^n, \dots, Y_{l_n}^n\}\}$ of FHDs implies the single FHD $X : \{Y_1, \dots, Y_k\}$ if and only if for each relation r with $X \cup \bigcup_{i=1}^k Y_i \cup \bigcup_{j=1}^n \left(X_j \cup \bigcup_{s=1}^{l_j} Y_s^j \right) \subseteq \text{Dom}(r)$ the FHD $X : \{Y_1, \dots, Y_k\}$ is satisfied by r whenever r already satisfies all FHDs in Σ .* \square

In this definition, the underlying relation schema is left undetermined. The only requirement is that the FHDs must apply to the relations.

Fact 1. *Let R be some relation schema and let $X \cup \bigcup_{i=1}^k Y_i \cup \bigcup_{j=1}^n \left(X_j \cup \bigcup_{s=1}^{l_j} Y_s^j \right) \subseteq R$. Then $\Sigma = \{X_1 : \{Y_1^1, \dots, Y_{l_1}^1\}, \dots, X_n : \{Y_1^n, \dots, Y_{l_n}^n\}\}$ R -implies $X : \{Y_1, \dots, Y_k\}$ whenever Σ implies $X : \{Y_1, \dots, Y_k\}$.*

The converse of Fact 1, however, is false as the following example shows.

Example 4.1. *Let Σ consist of the single FHD Title : {Actor, Feature}, and let R be the relation schema*

with the four attributes Title, Actor, Feature, Language. Then $\text{Title} : \{\text{Actor}, \text{Language}\}$ is R -implied by Σ by the soundness of the R -complementation rule. However, $\text{Title} : \{\text{Actor}, \text{Language}\}$ is not implied by Σ as the following counterexample relation r shows.

Title	Actor	Feature	Language	Crew
Musashi	T. Mifune	Trailer	English	H. Hinagaki
Musashi	T. Mifune	Trailer	Japanese	H. Hojo

While $r = r[\text{Title Actor}] \bowtie r[\text{Title Feature}] \bowtie r[\text{Title Language Crew}]$ we have $r \neq r[\text{Title Actor}] \bowtie r[\text{Title Language}] \bowtie r[\text{Title Feature Crew}]$. \square

An inference rule is called *sound* if the set of dependencies in the premise of the rule implies the dependency in the conclusion. A set \mathfrak{S} of inference rules is called *sound* for the implication of FHDs if and only if for every finite set Σ of FHDs we have $\Sigma_{\mathfrak{S}}^+ \subseteq \Sigma^* = \{\sigma \mid \Sigma \text{ implies } \sigma\}$. The set \mathfrak{S} is called *complete* for the implication of FHDs if and only if for every finite set Σ of FHDs we have $\Sigma^* \subseteq \Sigma_{\mathfrak{S}}^+$. An inference rule \mathfrak{R} is said to be *independent* from the set \mathfrak{S} if and only if there is some finite set $\Sigma \cup \{\sigma\}$ of FHDs such that $\sigma \notin \Sigma_{\mathfrak{S}}^+$, but $\sigma \in \Sigma_{\mathfrak{S} \cup \{\mathfrak{R}\}}^+$. A complete set \mathfrak{S} of inference rules is called *minimal* if and only if every inference rule \mathfrak{R} in \mathfrak{S} is independent from $\mathfrak{S} - \{\mathfrak{R}\}$. This means that no proper subset of \mathfrak{S} is still complete for the implication of FHDs.

While $\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{S}, \mathcal{O}, \mathcal{U}, \mathcal{D}, \mathcal{I}, \mathcal{I}_\emptyset, \mathcal{M}$ are all sound inference rules (since they are R -sound for all R), the R -axiom and the R -complementation rule \mathcal{C}_R are R -sound but not sound, see Example 4.1.

We will verify in this section that the set $\mathfrak{H}_C = \{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{S}, \mathcal{O}, \mathcal{M}\}$ is sound and complete for the implication of FHDs. That is, \mathfrak{H}_C can generate exactly the implications in an undetermined universe.

Lemma 4.1. *Let $\Sigma = \{X_1 : \{Y_1^1, \dots, Y_{l_1}^1\}, \dots, X_n : \{Y_1^n, \dots, Y_{l_n}^n\}\}$ be a finite set of FHDs. If $X : \{Y_1, \dots, Y_k\} \in \Sigma_{\mathfrak{H}_C}^+$, then for all $i = 1, \dots, k$ we have*

$$Y_i \subseteq \bigcup_{j=1}^n \bigcup_{s=1}^{l_j} Y_s^j.$$

Proof. We will show by induction on the length l of the inference $\gamma = [\sigma_1, \dots, \sigma_m]$ from Σ by \mathfrak{H}_C such that γ infers $\sigma_m = X : \{Y_1, \dots, Y_k\}$, then for all

$$i = 1, \dots, k \text{ we have } Y_i \subseteq \bigcup_{j=1}^n \bigcup_{s=1}^{l_j} Y_s^j.$$

Let $m = 1$. Then $X : \{Y_1, \dots, Y_k\} = \sigma_1 = \emptyset : \{\emptyset\}$ or $\sigma_1 \in \Sigma$. In each of these cases the desired property is indeed satisfied.

Let $m > 1$. One would need to consider six cases by which σ_m is inferred from $[\sigma_1, \dots, \sigma_{m-1}]$. We only consider the case where we obtain σ_m by applying the *transitivity rule* \mathcal{T} to the premises $\sigma_{i_1}, \sigma_{i_2}$ where $i_1, i_2 < m$. Then the last step of γ has the following form:

$$\frac{X' : \{Y'\}, X'Y' : \{Y'_1, \dots, Y'_{k-1}\}}{X' : \{Y'_1, \dots, Y'_{k-1}, Y'\}}.$$

The induction hypothesis shows $Y'_i \subseteq \bigcup_{j=1}^n \bigcup_{s=1}^{l_j} Y_s^j$ for

$$\text{all } i = 1, \dots, k-1 \text{ and } Y' \subseteq \bigcup_{j=1}^n \bigcup_{s=1}^{l_j} Y_s^j. \text{ This already}$$

proves the condition in this case. The remaining cases are similar. \square

Lemma 4.2. *Let $\Sigma = \{X_1 : \{Y_1^1, \dots, Y_{l_1}^1\}, \dots, X_n : \{Y_1^n, \dots, Y_{l_n}^n\}\}$ be a finite set of FHDs. Let $W :=$*

$\bigcup_{j=1}^n (X_j \cup \bigcup_{s=1}^{l_j} Y_s^j)$. If $X : \{Y_1, \dots, Y_k\} \in \Sigma_{\mathfrak{H}_C}^+$, then there is an inference $\gamma = [\sigma_1, \dots, \sigma_m]$ of $X : \{Y_1, \dots, Y_k\}$ from Σ by \mathfrak{H}_C such that any attribute occurring in $\sigma_1, \dots, \sigma_{m-1}$ is an element of W . \square

Theorem 4.1. *The set $\mathfrak{H}_C = \{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{S}, \mathcal{M}\}$ is sound and complete for the implication of FHDs.*

Proof. Let $\Sigma = \{X_1 : \{Y_1^1, \dots, Y_{l_1}^1\}, \dots, X_n : \{Y_1^n, \dots, Y_{l_n}^n\}\}$ be a finite set of FHDs, and let $X : \{Y_1, \dots, Y_k\}$ be an FHD. We need to prove that

$$\Sigma \text{ implies } X : \{Y_1, \dots, Y_k\} \text{ if and only if } X : \{Y_1, \dots, Y_k\} \in \Sigma_{\mathfrak{H}_C}^+ \quad (4.1)$$

For convenience let us define $T := X \cup \bigcup_{i=1}^k Y_i \cup$

$$\bigcup_{j=1}^n \left(X_j \cup \bigcup_{s=1}^{l_j} Y_s^j \right).$$

In order to prove the soundness of \mathfrak{H}_C we assume $X : \{Y_1, \dots, Y_k\} \in \Sigma_{\mathfrak{H}_C}^+$. Let r be any relation such that $T \subseteq \text{Dom}(r)$ and such that all FHDs $X_i : \{Y_1^i, \dots, Y_{l_i}^i\} \in \Sigma$ are satisfied by r . Then we need to show that r also satisfies $X : \{Y_1, \dots, Y_k\}$.

According to Lemma 4.2 there is an inference γ of

$$X : \{Y_1, \dots, Y_k\} \text{ from } \Sigma \text{ by } \mathfrak{H}_C \text{ such that } R \cup \bigcup_{i=1}^t S_i \subseteq$$

$T \subseteq \text{Dom}(r)$ for each FHD $R : \{S_1, \dots, S_t\}$ occurring in γ . Since each rule of \mathfrak{H}_C is sound we can conclude by induction that each FHD occurring in γ is satisfied by r . In particular, r satisfies also $X : \{Y_1, \dots, Y_k\}$.

In order to prove the completeness of \mathfrak{H}_C we assume $X : \{Y_1, \dots, Y_k\} \notin \Sigma_{\mathfrak{H}_C}^+$. Let $R \subseteq \mathfrak{A}$ be a finite set of attributes such that T is a proper subset of R , i.e. $T \subset R$.

Then $R - XY_1 \dots Y_k$ is not a subset of T . Hence, by Lemma 4.1

$$X : \{Y_1, \dots, Y_{i-1}, Y_{i+1}, \dots, Y_k, R - XY_1 \dots Y_k\} \notin \Sigma_{\mathfrak{H}_C}^+$$

for all $i = 1, \dots, k$. Now from $X : \{Y_1, \dots, Y_k\} \notin \Sigma_{\mathfrak{H}_C}^+$ and

$$X : \{Y_1, \dots, Y_{i-1}, Y_{i+1}, \dots, Y_k, R - XY_1 \dots Y_k\} \notin \Sigma_{\mathfrak{H}_C}^+$$

for all $i = 1, \dots, k$ we conclude that

$$X : \{Y_1, \dots, Y_k\} \notin \Sigma_{R\mathfrak{H}_C}^+$$

by Corollary 3.2.

Since $R\mathfrak{H}_C$ is complete for the R -implication of FHDs it follows that Σ does not R -imply $X : \{Y_1, \dots, Y_k\}$. Consequently, Σ does not imply $X : \{Y_1, \dots, Y_k\}$ by Fact 1. \square

The following lemmata show that the Boolean rules are indeed derivable from \mathfrak{H}_C .

Lemma 4.3. *The union rule \mathcal{U} is derivable from $\{\mathcal{A}, \mathcal{T}, \mathcal{M}\}$.*

Proof.

$$\frac{X : \{Y_1, \dots, Y_k\}}{X : \{Z\} \quad \frac{XZ : \{Y_1 - Z, \dots, Y_k - Z\}}{X : \{Y_1 - Z, \dots, Y_k - Z, Z\}}^{\mathcal{A}} \quad \frac{X : \{Y_1 - Z, \dots, Y_k - Z, Z\}}{X : \{Y_1 - Z, \dots, Y_{k-1} - Z, Y_k Z\}}^{\mathcal{M}} \quad \mathcal{T}$$

\square

Lemma 4.4. *The intersection rule \mathcal{I} is derivable from $\{\mathcal{A}, \mathcal{T}, \mathcal{M}, \mathcal{O}, \mathcal{S}\}$.* \square

Lemma 4.5. *The difference rule \mathcal{D} is derivable from $\{\mathcal{A}, \mathcal{T}, \mathcal{M}, \mathcal{O}\}$.* \square

5 The minimality of \mathfrak{H}_C and $R\mathfrak{H}_C$

It is now the goal to demonstrate the minimality of $\mathfrak{H}_C = \{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{S}, \mathcal{M}\}$ for the implication of FHDs. As a consequence the system $R\mathfrak{H}_C$ is minimal for the R -implication of FHDs in the sense that none of its subsets is still both complete and complementary for the R -implication of FHDs.

For the following independence proof sketches one can utilise Lemma 4.1 which restricts the number of possible FHDs that can be inferred from Σ by \mathfrak{H}_C .

Lemma 5.1. *The empty-set-axiom R_\emptyset is independent from $\{\mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{S}, \mathcal{M}\}$.*

Proof. Let $\Sigma = \emptyset$ and $\sigma = \emptyset : \{\emptyset\}$. Then $\sigma \notin \Sigma_{\{\mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{S}, \mathcal{M}\}}^+$, but $\sigma \in \Sigma_{\mathfrak{H}_C}^+$. \square

Lemma 5.2. *The augmentation rule \mathcal{A} is independent from $\{\mathcal{R}_\emptyset, \mathcal{T}, \mathcal{O}, \mathcal{S}, \mathcal{M}\}$.*

Proof. Let $\Sigma = \emptyset$ and $\sigma = A : \{\emptyset\}$. One can show that $\sigma \notin \Sigma_{\{\mathcal{R}_\emptyset, \mathcal{T}, \mathcal{O}, \mathcal{S}, \mathcal{M}\}}^+ = \{\emptyset : \{\emptyset\}\}$ but $\sigma \in \Sigma_{\mathfrak{H}_C}^+$. \square

Lemma 5.3. *The transitivity rule \mathcal{T} is independent from $\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{O}, \mathcal{S}, \mathcal{M}\}$.*

Proof. In this case we choose $\Sigma = \{A : \{B\}, AB : \{C\}\}$, and $\sigma = A : \{C\}$. One can show that $\sigma \notin \Sigma_{\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{O}, \mathcal{S}, \mathcal{M}\}}^+$ but $\sigma \in \Sigma_{\mathfrak{H}_C}^+$. \square

Lemma 5.4. *The omission rule \mathcal{O} is independent from $\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{S}, \mathcal{M}\}$.*

Proof. We choose $\Sigma = \{\emptyset : \{A, B\}\}$, and $\sigma = \emptyset : \{A\}$. One can show that $\sigma \notin \Sigma_{\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{S}, \mathcal{M}\}}^+$ but $\sigma \in \Sigma_{\mathfrak{H}_C}^+$. \square

Lemma 5.5. *The subset rule \mathcal{S} is independent from $\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{M}\}$.*

Proof. We choose $\Sigma = \{\emptyset : \{AB\}, C : \{B\}\}$ and $\sigma = \emptyset : \{B\}$. One can show that $\sigma \notin \Sigma_{\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{M}\}}^+$ but $\sigma \in \Sigma_{\mathfrak{H}_C}^+$. \square

Lemma 5.6. *The merging rule \mathcal{M} is independent from $\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{S}\}$.*

Proof. Let $\Sigma = \{\emptyset : \{A, B\}\}$ and $\sigma = \emptyset : \{AB\}$. One can show that $\sigma \notin \Sigma_{\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{S}\}}^+$ but $\sigma \in \Sigma_{\mathfrak{H}_C}^+$. \square

Theorem 5.1. *The set $\mathfrak{H}_C = \{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{S}, \mathcal{M}\}$ is minimal for the implication of FHDs.* \square

Since the R -complementation rule \mathcal{C}_R is independent from the rules in \mathfrak{H}_C Theorem 5.1 implies that none of the proper subsets of $R\mathfrak{H}_C$ can still be complete and complementary for the R -implication of FHDs.

Corollary 5.1. *None of the proper subsets of $R\mathfrak{H}_C = \{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{S}, \mathcal{M}, \mathcal{C}_R\}$ is still both complete and complementary for R -implication of FHDs.* \square

Finally, we show the independence of the *union rule* \mathcal{U} from $\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}\}$. This completes Example 3.1.

Lemma 5.7. *The union rule \mathcal{U} is independent from $\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}\}$.*

Proof. Let $\Sigma = \{\emptyset : \{A\}, \emptyset : \{B\}\}$ and $\sigma = \emptyset : \{AB\}$. The closure $\Sigma_{\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}\}}^+$ can be obtained as follows. The *empty-set-axiom* \mathcal{R}_\emptyset yields the FHD $\emptyset : \{\emptyset\}$. A subsequent applications of the *augmentation rule* \mathcal{A} allows us to derive the \emptyset -column. One may then enter the FHDs from Σ and apply the *augmentation rule* \mathcal{A} subsequently to derive the $\{A\}$ -column and $\{B\}$ -column. One can then apply the *transitivity rule* \mathcal{T} to $\emptyset : \{A\}$ and $A : \{B\}$ to infer $\emptyset : \{A, B\}$. This set is closed under derivation using $\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}\}$.

	$\{\emptyset\}$	$\{A\}$	$\{B\}$	$\{AB\}$	$\{A, B\}$
\emptyset	\times	\times	\times		\times
A	\times	\star	\times	\star	\star
B	\times	\times	\star	\star	\star
AB	\times	\star	\star	\star	\star

It follows that $\sigma \notin \Sigma_{\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}\}}^+$ but $\sigma \in \Sigma_{\{\mathcal{R}_\emptyset, \mathcal{A}, \mathcal{T}, \mathcal{O}, \mathcal{U}\}}^+$. \square

6 Future Work

We conclude this paper by listing some related problems that warrant further research.

The *subset rule* plays a key role in achieving complementarity for MVDs and FHDs. It would be interesting to see whether there are any complete sets of inference rules for the implication of MVDs (FHDs) in undetermined universes that do not feature the subset rule.

While FDs and MVDs have been investigated before and an axiomatisation is well-known for the class of both types of dependencies in fixed universes (Beeri et al. 1977) the combined class of FDs and FHDs should also be studied in both fixed and undetermined universes.

According to (Link 2006a) there seems to be a trade-off between minimality and complementarity for MVDs. That is, so far no minimal complete set of inference rules for the R -implication of MVDs has been identified that is also complementary. The question is whether there is any such system.

MVDs have been studied in the presence of null values, for instance with interpretation *no information* (Lien 1982, Link 2006b). Interestingly, the soundness of the *transitivity rule* fails when database relations are allowed to be incomplete. FHDs should therefore also be studied in the presence of null values. An interesting approach to incomplete data has been applied to the class of functional dependencies (Levene & Loizou 1998). There, a possible world semantics is used to explore all possible extensions of an incomplete database to a complete database. While weak FDs are satisfied by some possible world, strong FDs must be satisfied by all possible worlds. It would be interesting to generalise this work to MVDs (FHDs).

An open problem is the lack of a synthesis algorithm for MVDs that would extend the well-known synthesis algorithm for the class of FDs (Bernstein 1976, Biskup, Dayal & Bernstein 1979). The notion of MVD implication in undetermined universes provides an alternative basis for the formulation of such an algorithm.

References

- Arenas, M. & Libkin, L. (2004), 'A normal form for XML documents', *ToDS* **29**(1), 195–232.
- Armstrong, W. W. (1974), 'Dependency structures of database relationships', *Information Processing* **74**, 580–583.
- Armstrong, W. W., Nakamura, Y. & Rudnicki, P. (2002), 'Armstrong's axioms', *Journal of formalized Mathematics* **14**.
- Beeri, C. (1980), 'On the membership problem for functional and multivalued dependencies in relational databases', *ToDS* **5**(3), 241–259.
- Beeri, C. & Bernstein, P. A. (1979), 'Computational problems related to the design of normal form relational schemata', *ToDS* **4**(1), 30–59.
- Beeri, C., Fagin, R. & Howard, J. H. (1977), A complete axiomatization for functional and multivalued dependencies in database relations, in 'SIGMOD', ACM, pp. 47–61.
- Bernstein, P. (1976), 'Synthesizing third normal form relations from functional dependencies', *ToDS* **1**(4), 277–298.
- Bernstein, P. A. & Goodman, N. (1980), What does Boyce-Codd normal form do?, in 'VLDB', IEEE Computer Society, pp. 245–259.
- Biskup, J. (1978), 'On the complementation rule for multivalued dependencies in database relations', *Acta Informatica* **10**(3), 297–305.
- Biskup, J. (1980), 'Inferences of multivalued dependencies in fixed and undetermined universes', *TCS* **10**(1), 93–106.
- Biskup, J., Dayal, U. & Bernstein, P. (1979), Synthesizing independent database schemas, in 'SIGMOD', pp. 143–151.
- Codd, E. F. (1970), 'A relational model of data for large shared data banks', *Commun. ACM* **13**(6), 377–387.
- Codd, E. F. (1972), Further normalization of the database relational model, in 'Courant Computer Science Symposia 6: Data Base Systems', Prentice-Hall, pp. 33–64.
- Delobel, C. (1978), 'Normalisation and hierarchical dependencies in the relational data model', *ToDS* **3**(3), 201–222.
- Fagin, R. (1977), 'Multivalued dependencies and a new normal form for relational databases', *ToDS* **2**(3), 262–278.
- Fagin, R. & Vardi, M. Y. (1986), The theory of data dependencies: a survey, in 'Mathematics of Information Processing: Proceedings of Symposia in Applied Mathematics', American Mathematical Society, pp. 19–71.
- Fischer, P. C., Saxton, L. V., Thomas, S. J. & Van Gucht, D. (1985), 'Interactions between dependencies and nested relational structures', *J. Comput. Syst. Sci.* **31**(3), 343–354.
- Graetzer, G. (1998), *General Lattice Theory*, Birkhauser.
- Hara, C. & Davidson, S. (1999), Reasoning about nested functional dependencies, in 'PoDS', ACM, pp. 91–100.
- Hartmann, S. & Link, S. (2004), Multi-valued dependencies in the presence of lists, in 'PoDS', ACM, pp. 330–341.
- Hartmann, S. & Link, S. (2006), 'On a problem of Fagin concerning multivalued dependencies in relational databases', *TCS* **353**(1-3), 53–62.
- Hartmann, S., Link, S. & Schewe, K.-D. (2006), 'Functional and multivalued dependencies in nested databases generated by record and list constructor', *AMAI* **46**(1-2), 114–164.
- Levene, M. & Loizou, G. (1998), 'Axiomatisation of functional dependencies in incomplete relations', *TCS* **206**(1-2), 283–300.
- Lien, Y. E. (1982), 'On the equivalence of data models', *Journal of the ACM* **29**(2), 333–363.
- Link, S. (2006a), On multivalued dependencies in fixed and undetermined universes, in 'FoIKS', number 3861 in 'Lecture Notes in Computer Science', pp. 257–276.
- Link, S. (2006b), On the logical implication of multivalued dependencies with null values, in 'Twelfth Computing: The Australasian Theory Symposium', number 51 in 'CRPIT', pp. 113–122.
- Mendelzon, A. (1979), 'On axiomatising multivalued dependencies in relational databases', *J. ACM* **26**(1), 37–44.
- Tari, Z., Stokes, J. & Spaccapietra, S. (1997), 'Object normal forms and dependency constraints for object-oriented schemata', *ToDS* **22**, 513–569.
- Thalheim, B. (1991), *Dependencies in Relational Databases*, Teubner-Verlag.
- Thalheim, B. (2000), *Entity-Relationship Modeling: Foundations of Database Technology*, Springer.
- Thalheim, B. (2003), Conceptual treatment of multivalued dependencies, in 'Conceptual Modeling - ER 2003, Proceedings', number 2813 in 'Lecture Notes in Computer Science', Springer, pp. 363–375.
- Vardi, M. Y. (1987), Fundamentals of dependency theory, in E. Börger, ed., 'Trends in Theoretical Computer Science', Computer Science Press, pp. 171–224.
- Vincent, M. & Liu, J. (2003), Multivalued dependencies in XML, in 'BNCOD', number 2712 in 'Lecture Notes in Computer Science', Springer, pp. 4–18.
- Vincent, M., Liu, J. & Liu, C. (2003), A redundancy free 4NF for XML, in 'XMLSym', number 2824 in 'Lecture Notes in Computer Science', Springer, pp. 254–266.
- Weddell, G. (1992), 'Reasoning about functional dependencies generalized for semantic data models', *ToDS* **17**(1), 32–64.
- Wijzen, J. (1999), 'Temporal FDs on complex objects', *ToDS* **24**(1), 127–176.
- Zaniolo, C. (1976), Analysis and Design of Relational Schemata for Database Systems, PhD thesis, UCLA, Tech. Rep. UCLA-ENG-7769.

Domination Normal Form - Decomposing Relational Database Schemas

Henning Koehler

Massey University, Palmerston North, New Zealand
h.koehler@massey.ac.nz

Abstract

A common approach in designing relational databases is to start with a universal relation schema, which is then decomposed into multiple subschemas. A good choice of subschemas can be determined using integrity constraints defined on the schema, such as functional, multivalued or join dependencies.

In this paper we propose and analyze a new normal form based on the idea of minimizing overall storage space. This is in contrast to existing normal forms such as BCNF, 4NF or KCNF, which only characterize the absence of redundancy (and thus space-minimality) for a single schema.

1 Introduction

We begin by introducing some basic terms from relational database theory, followed by a description of the problem we are trying to solve.

1.1 Terminology

A *relation schema* $R = \{A_1, A_2, \dots, A_n\}$ is a set of *attributes*. A *relation* r over a schema R is a set of tuples (where each tuple represents one data item), and each element of the tuple corresponds to one attribute in R .

With each relation schema we associate a set Σ of integrity constraints, in particular *functional dependencies* (FD), *multivalued dependencies* (MVD) and *join dependencies* (JD). These restrict which relations over R we may store. We say that a set Σ of constraints over R *implies* a constraint c , written $\Sigma \models c$, if c holds on every relation r over R for which all constraints in Σ hold.

A FD on R is an expression of the form $X \rightarrow Y$ (read " X determines Y ") where X and Y are subsets of R . We say that a FD $X \rightarrow Y$ *holds* on a relation r over R if every pair of tuples in r that coincides on all attributes in X also coincides on all attributes in Y . We call a FD $X \rightarrow Y$ *trivial* if $Y \subseteq X$, or, equivalently, if it holds on every relation. A set $X \subseteq R$ is a *key* of R if Σ implies $X \rightarrow R$.

For a set $X \subseteq R$ we denote the *projection* of r onto the attributes in X by $r[X]$. The *join* of two relations $r[X]$ and $r[Y]$ is a relation on $X \cup Y$:

$$r[X] \bowtie r[Y] := \left\{ t \mid \begin{array}{l} \exists t_1 \in r[X], t_2 \in r[Y]. \\ t[X] = t_1 \wedge t[Y] = t_2 \end{array} \right\}$$

A join dependency on R is an expression of the form $\bowtie [R_1, \dots, R_n]$ where the R_i are subsets of R with $\bigcup R_i = R$. We say that the JD $\bowtie [R_1, \dots, R_n]$ holds on r if the decomposition $\{R_1, \dots, R_n\}$ is loss-less for r , i.e. if

$$r[R_1] \bowtie \dots \bowtie r[R_n] = r$$

A multivalued dependency is a join dependency $\bowtie [R_1, R_2]$ with only two subschemas. It is usually written as $X \twoheadrightarrow Y$ where $X = R_1 \cap R_2$ and $Y = R_1 \setminus R_2$ or (equivalently) $Y = R_2 \setminus R_1$.

1.2 Problems with Existing Normal Forms

When presented with a relation schema R and a set of constraints Σ on it, a designer must decide whether to store all data in one single relation on R , or to decompose R into multiple subschemas. To aid in this task, normal forms are used which characterize "good" solutions.

Many normal forms proposed so far, such as BCNF, 4NF or KCNF, characterize the absence of redundancy. This is desirable for several reasons, foremost the avoidance of update anomalies [4].

However, these normal forms have significant drawbacks. First, they only consider a single relation schema, instead of considering all schemas in a decomposition together.

Definition 1. For a relation r over R and a decomposition $\mathcal{R} = \{R_1, \dots, R_n\}$ of $R = \bigcup R_j$ we denote the decomposition of r by \mathcal{R} as

$$r[\mathcal{R}] := \{r[R_1], \dots, r[R_n]\}$$

where $r[R_j]$ is the projection of r onto the attributes in R_j . When talking about the tuples in $r[\mathcal{R}]$ containing an attribute A , we shall mean the tuples from relations $R_j \in \mathcal{R}$ with $A \in R_j$.

The common generalization to multiple schemas is that the whole schema collection is in that normal form, if every schema taken individually is. But this means that those normal forms cannot capture redundancy which exists across multiple relations. As a trivial example, we can duplicate a schema. Clearly the extra schema is then superfluous in the whole schema collection, but each schema taken individually may still be redundancy free.

Example 1. Let $R = ABCD$ be a schema with constraints $\Sigma = \{AB \rightarrow CD, CD \rightarrow B\}$. Then R is not in BCNF, but has a dependency preserving BCNF decomposition into the subschemas ABC, ABD, BCD .

For any instance r of R , the projections of r onto the schemas ABC and ABD together already take up more space than the original relation r : no tuples are lost in the projection since AB is a key, and

the attributes A and B are stored twice. While we have not defined what redundancy means for multiple schemas, it seems intuitively clear that this decomposition should not be called "redundancy free". From a storage space point-of-view, it is clearly less desirable than the original schema R .

The second big problem is that dependency preserving decompositions into these normal forms do not always exist [2]. Thus, when faced with such a case, a designer must either accept the loss of some dependencies, or cannot use the normal form in question.

We believe that what a normal form should do, is characterize "good" representations (i.e. decompositions) of a schema, in such a way that a "good" representation does always exist. In the following we will propose a normal form which meets this criterion.

2 Minimization as Normal Form

The approach we suggest is the following: among a set of suitable decompositions (e.g. the set of all lossless, or lossless and dependency preserving decompositions), characterize the "best" ones. We do so by defining an order on the decompositions, such that the "best" decompositions are the minimal ones with respect to that order.

This leaves the question of when to call one decomposition better than another one. The motivation for many normal forms proposed so far has been the elimination of redundancy (and with it, the absence of update-anomalies). This may suggest to define a quantitative measure of redundancy over multiple schemas, as has been done in [1].

We take a slightly different route here: instead of trying to minimize redundancy, we try to minimize the size of instances. Intuitively this should lead to similar results, but measures for size appear easier to construct. In the following we will define and motivate three different orders on decompositions, all of which intuitively measure the size.

2.1 Ordering by Size of Instances

Our first approach measures the space required to store an instance. For that we need to know for each element of a domain how much storage space it requires. We represent this knowledge by associating with each domain Dom a size function

$$size : Dom \rightarrow Int$$

Consider e.g. the following domains:

- $STRING$ containing strings of arbitrary length
- $STRING[40]$ containing strings of length up to 40
- INT containing arbitrarily large integers
- $INT(64)$ containing all 64-bit integers
- $BOOLEAN$ containing the values $TRUE$ and $FALSE$

A realistic measure for the size of a string might be its length, the size of an integer i might be defined as $\log(i)$, and the size of $TRUE$ and $FALSE$ might be one.

We shall assume that all domains are infinite, and that the size functions on them are positive and unbounded, i.e. can grow arbitrarily large. This can be justified as follows: For any infinite domain a bounded size function is not realistic, since we can store only a finite number of element when constricted to a fixed

amount of space. While not all domains are truly infinite, they often contain far more elements than the number of subschemas in a typical decomposition (e.g. 256^{40} for $STRING[40]$ or 2^{64} for $INT(64)$). Treating those domains as infinite will allow us to draw a sharp boundary between small (bounded) increases in size from duplicated attributes on one hand, and potentially large (unbounded) increases in size from instances with large numbers of tuples on the other.

We note that this argument fails for domains such as $BOOLEAN$, and that a constant size function may be more realistic for domains such as $STRING[40]$ or $INT(64)$. We won't consider finite domains or constant size functions in this paper though.

As it will turn out, the assumptions about infinite domains and unbounded size functions are all we need to characterize our new normal form, i.e. we do not require detailed knowledge about the actual size functions.

Definition 2. Let $R = \{A_1, \dots, A_k\}$ be a schema. For a finite relation r over R we define the size of r as

$$size(r) := \sum_{t \in r} \sum_{i=1}^k size(\pi_{A_i}(t))$$

We then define the size of the decomposition of r by $\mathcal{R} = \{R_1, \dots, R_n\}$ of R as

$$size(r[\mathcal{R}]) := \sum_{j=1}^n size(r[R_j])$$

While this gives us a suitable definition of size for any instance, we wish to compare schemas w.r.t. the size of all valid instances. If for every valid instance r on R a decomposition \mathcal{R}_1 requires no more storage space than a decomposition \mathcal{R}_2 , then $\mathcal{R}_1 \leq \mathcal{R}_2$ should certainly hold, indicating that \mathcal{R}_1 is "smaller" or "better" than \mathcal{R}_2 . Recall that we shall only consider suitable decompositions, in particular lossless or lossless and dependency preserving ones.

This alone, however, is not sufficient to characterize good decompositions: for an instance r containing only a single element, the trivial decomposition $\{R\}$ requires less storage space than any other lossless decomposition, as those typically need to duplicate some attributes. It would be hard to argue though that decomposition is never necessary. So how can we distinguish decompositions finer, based on the size of instances?

Example 2. Let $R = ABC$ and $\Sigma = \{B \rightarrow C\}$. R can be faithfully decomposed into $\mathcal{R} = \{AB, BC\}$. Clearly every relation r decomposed by \mathcal{R} (which is a set of relations) is at most twice as large as r . On the other hand, for every natural number k we can construct a relation

$$r = \begin{array}{|c|c|c|} \hline A & B & C \\ \hline 1 & 1 & \text{"a very long string"} \\ \hline 2 & 1 & \text{"a very long string"} \\ \hline \vdots & \vdots & \vdots \\ \hline k+1 & 1 & \text{"a very long string"} \\ \hline \end{array}$$

which is more than k times larger than in decomposed form:

A	B
1	1
2	1
⋮	⋮
k+1	1

B	C
1	"a very long string"

This observation motivates the following definitions, which compare decompositions similarly to the "big- O " comparison (e.g. $3x^2 + x \in O(x^2)$) from complexity theory.

Definition 3. Let R be a schema with constraints Σ and $\mathcal{R}_1, \mathcal{R}_2$ be decompositions of R . We say that \mathcal{R}_1 *c-dominates* \mathcal{R}_2 (where "c" stands for complexity) if there exists a constant k such that for all finite relations r over R that satisfy Σ we have

$$\text{size}(r[\mathcal{R}_1]) \leq k \cdot \text{size}(r[\mathcal{R}_2])$$

We further say that \mathcal{R}_1 *dominates* \mathcal{R}_2 if the above relation holds for $k = 1$. We abbreviate c-domination and domination as $\mathcal{R}_1 \leq_c \mathcal{R}_2$ and $\mathcal{R}_1 \leq \mathcal{R}_2$, respectively. We say that \mathcal{R}_1 *strictly (c-)dominates* \mathcal{R}_2 , written $\mathcal{R}_1 <_{(c)} \mathcal{R}_2$, if \mathcal{R}_1 (c-)dominates \mathcal{R}_2 but not vice-versa.

It is easy to see that both domination and c-domination are reflexive and transitive, and thus are pre-orders. Clearly domination implies c-domination.

Proposition 1. Let $\mathcal{R}_1, \mathcal{R}_2$ be decompositions of R . If \mathcal{R}_1 dominates \mathcal{R}_2 then \mathcal{R}_1 c-dominates \mathcal{R}_2 .

Note however that strict domination does *not* imply strict c-domination. Going back to example 2, we see that R and \mathcal{R} are incomparable w.r.t. domination, but \mathcal{R} strictly c-dominates R . In example 1 the original schema $ABCD$ strictly dominates the decomposition $\{ABC, ABD\}$, but both decompositions are equivalent w.r.t. c-domination. Sometimes both criteria, domination and c-domination, are needed to characterize the best decomposition for a schema:

Example 3. Let $R = ABCDE$ be a schema with constraints $\Sigma = \{AB \rightarrow CD, B \rightarrow E\}$. Then the decomposition $\mathcal{R}_1 = \{ABC, ABD, BE\}$ is minimal w.r.t. c-domination but strictly dominated by $\mathcal{R}_2 = \{ABCD, BE\}$. The trivial decomposition $\{R\}$ is minimal w.r.t. domination but strictly c-dominated by both \mathcal{R}_1 and \mathcal{R}_2 .

We are now ready to define our normal form based on the idea of minimizing storage space.

Definition 4. Let R be a schema with constraints Σ and \mathcal{R} be a decomposition of R . We say that \mathcal{R} is in *domination normal form* (DNF) if

- (i) \mathcal{R} is minimal w.r.t. domination, and
- (ii) \mathcal{R} is minimal w.r.t. c-domination

with minimal meaning that no strictly smaller decomposition exists among a given set of 'suitable' decompositions.

Note that this definition depends on the choice which decompositions we consider 'suitable'. We will investigate two different cases (though other choices might be of interest as well): the set of all lossless, and the set of all lossless and dependency preserving decompositions. In each case, we effectively obtain a different DNF.

As the number of suitable decompositions of a given schema is finite, there must exist a decomposition among them which is minimal w.r.t. domination, as well as a (possibly different) schema which is minimal w.r.t. c-domination. It is however not clear yet whether a decomposition into DNF always exists, i.e. one which is minimal w.r.t. both criteria at once. We will show this next.

Theorem 2. Every schema has a decomposition into DNF.

Proof. We use the fact that domination implies c-domination. The c-domination pre-order induces a partition of all the (suitable) decompositions of R into equivalence classes and defines a partial order on those equivalence classes. Let EQ be a minimal equivalence class w.r.t. that order. Choose \mathcal{R} to be minimal w.r.t. domination among the decompositions in EQ . We claim that \mathcal{R} is minimal w.r.t. domination among *all* decompositions of R , and thus in DNF. Let \mathcal{R}' be any decomposition with $\mathcal{R}' \leq \mathcal{R}$. Then $\mathcal{R}' \leq_c \mathcal{R}$, and since \mathcal{R} is minimal w.r.t. c-domination, $\mathcal{R}' \in EQ$. But \mathcal{R} is also minimal w.r.t. domination in EQ , and thus $\mathcal{R} \leq \mathcal{R}'$. Thus no decomposition \mathcal{R}' strictly dominates \mathcal{R} . \square

2.2 Ordering by Attribute Count of Instances

Instead of measuring the total size of instances, we could count the number of tuples an attribute appears in. This gives us domination and c-domination pre-orders for each attribute, and we can then combine these to get another pair of orderings for decompositions.

Definition 5. Let $R = \{A_1, \dots, A_k\}$ be a schema. For a finite relation r over R and an attribute A we define the *count* of A on r as

$$\text{count}_A(r) := \begin{cases} |r| & \text{if } A \in R \\ 0 & \text{if } A \notin R \end{cases}$$

where $|r|$ denotes the number of tuples in r . We then define the count of A on r decomposed by a decomposition $\mathcal{R} = \{R_1, \dots, R_n\}$ of R as

$$\text{count}_A(r[\mathcal{R}]) := \sum_{j=1}^n \text{count}_A(r[R_j])$$

Definition 6. Let R be a schema with FDs F , A an attribute and $\mathcal{R}_1, \mathcal{R}_2$ decompositions of R . We say that \mathcal{R}_1 *c-dominates* \mathcal{R}_2 w.r.t. A if there exists a constant k such that for all finite relations r over R that satisfy F we have

$$\text{count}_A(r[\mathcal{R}_1]) \leq k \cdot \text{count}_A(r[\mathcal{R}_2])$$

We further say that \mathcal{R}_1 *dominates* \mathcal{R}_2 w.r.t. A if the above relation holds for $k = 1$.

Thus, for each attribute A , we get a c-domination and domination pre-orders. We shall combine those pre-orders by intersection.

Definition 7. Let R be a schema and $\mathcal{R}_1, \mathcal{R}_2$ decomposition of R . We say that \mathcal{R}_1 $\left\{ \begin{array}{l} \text{dominates} \\ \text{c-dominates} \end{array} \right\}$ \mathcal{R}_2 w.r.t. attribute count if \mathcal{R}_1 $\left\{ \begin{array}{l} \text{dominates} \\ \text{c-dominates} \end{array} \right\}$ \mathcal{R}_2 w.r.t. every attribute.

It will turn out that domination and c-domination w.r.t. attribute count and w.r.t. size are exactly the same orders.

2.3 Ordering by Containing Schema Closures

The previous two order pairs introduced are defined by considering all valid instances. This is not very practical if we wish to actually decide for two given decompositions whether one (c-) dominates the other. We shall therefore give a third pair of orders, which is defined by considering only the decompositions, rather than instances on them.

The approach we use is similar to attribute counting. The count of an attribute depends on the set of schemas it lies in. While it also depends on the instance r , it is easy to show that the number of tuples in $r[R_j]$ is determined by the closure $\overline{R_j}$ of R_j (and r). Recall that the closure $\overline{R_j}$ of R_j under Σ is

$$\overline{R_j} := \{A \in R \mid \Sigma \models R_j \rightarrow A\}$$

where Σ is an arbitrary set of constraints on R (we shall mainly be interested in functional, multi-valued and join dependencies).

Lemma 3. *Let R be a schema with constraints Σ , and $X \subseteq R$. Then for all relations r on R we have $|r[X]| = |r[\overline{X}]|$.*

Proof. We can obtain $r[X]$ by projecting from $r[\overline{X}]$. We could only lose tuples if $r[\overline{X}]$ contains different tuples which are identical on X . But this can't happen since X functionally determines \overline{X} . \square

This motivates the following definitions.

Definition 8. Let R be a schema with constraints Σ , and \mathcal{R} a decomposition of R . Then for any attribute $A \in R$ we define the *containing schema closures* (CSC) of A in \mathcal{R} as the *multiset*

$$CSC_A(\mathcal{R}) = \{\overline{R_j} \mid A \in R_j \in \mathcal{R}\}$$

Note that it is necessary to use multisets rather than sets to obtain the correct attribute count.

Example 4. Consider again the schema $R = ABCDE$ with constraints $\Sigma = \{AB \rightarrow CD, B \rightarrow E\}$ from example 3, and the decompositions

$$\mathcal{R}_1 = \{ABC, ABD, BE\}$$

$$\mathcal{R}_2 = \{ABCD, BE\}$$

They produce the multisets

$$CSC_A(\mathcal{R}_1) = \{ABCDE, ABCDE\}$$

$$CSC_A(\mathcal{R}_2) = \{ABCDE\}$$

which indicate that, for any relation r on R , the attribute A appears in twice as many tuples in $r[\mathcal{R}_1]$ as in $r[\mathcal{R}_2]$. Using sets would hide this difference.

We shall now compare decompositions using the respective CSCs of all attributes. For that we need mappings between multi-sets. We shall allow different instances of the same value in the source domain to map to different values, and call a mapping injective if a value in the target domain is mapped to at most as often as it occurs in the target domain.

Definition 9. Let M_1, M_2 be two multisets¹ of (attribute) sets. We say that

- (i) M_1 *weakly inclusion-dominates* M_2 if there exists a mapping $f : M_1 \rightarrow M_2$ with $e \subseteq f(e)$ for all $e \in M_1$.
- (ii) M_1 *strongly inclusion-dominates* M_2 if there exists an injective mapping $f : M_1 \rightarrow M_2$ with $e \subseteq f(e)$ for all $e \in M_1$.

Definition 10. Let $\mathcal{R}_1, \mathcal{R}_2$ be two decompositions of R . We say that \mathcal{R}_1 *weakly/strongly csc-dominates* \mathcal{R}_2 if for all attributes $A \in R$ we have that $CSC_A(\mathcal{R}_1)$ weakly/strongly inclusion-dominates $CSC_A(\mathcal{R}_2)$.

It will turn out that weak csc-domination implies c-domination, and that strong csc-domination implies domination. While the opposite does not hold for arbitrary types of constraints, we will be able to show that it holds for sets of functional dependencies, and in the case of weak csc-domination/c-domination also for multi-valued and join dependencies.

¹Note that for definition (i) we do not really need multi-sets.

3 Equivalence of Orderings

We will show that, if the only constraints on the schema are functional, multi-valued and join dependencies, then all three order pairs defined in section 2 are identical (with one possible exception which we will discuss later). As multi-valued dependencies are just a special case of join dependencies, it suffices to consider only functional and join dependencies. We will assume throughout this section that functional and join dependencies are the only types of integrity constraints occurring.

3.1 Size vs. Attribute Count

We start by showing that the orders defined by size and attribute count are identical. Recall that we assume that all domains are infinite, and that the size functions associated with them are positive and unbounded.

Lemma 4. *Let R be a schema with constraints Σ , and $\mathcal{R}_1, \mathcal{R}_2$ be decompositions of R . If \mathcal{R}_1 does not (c-)dominate \mathcal{R}_2 w.r.t. attribute count, then \mathcal{R}_1 does not (c-)dominate \mathcal{R}_2 w.r.t. size.*

Proof. If \mathcal{R}_1 does not c-dominate \mathcal{R}_2 w.r.t. attribute count, then for every integer k there exists a relation r and an attribute A with

$$\text{count}_A(r[\mathcal{R}_1]) > k \cdot \text{count}_A(r[\mathcal{R}_2])$$

For non-domination such r and A only need to exist for $k = 1$. For each k and associated r and A , we shall construct a relation r' for which

$$\text{size}(r'[\mathcal{R}_1]) > k \cdot \text{size}(r'[\mathcal{R}_2])$$

holds. This shows non-(c-)domination w.r.t. size, thus proving the lemma.

The construction works as follows. Since the relations in $r[\mathcal{R}_1]$ with attribute A contain more than k times as many tuples as those in $r[\mathcal{R}_2]$, there must be an attribute value v_A for A which appears more than k times as often in $r[\mathcal{R}_1]$ as in $r[\mathcal{R}_2]$.

We construct r' from r by substituting every occurrence of v_A by a new value v'_A which doesn't appear in r . As Σ contains only functional and join dependencies, these constraints still hold for r' . Let o_1, o_2 be the number of occurrences of v_A in $r[\mathcal{R}_1], r[\mathcal{R}_2]$. We choose v'_A sufficiently large, i.e. such that

$$\text{size}(v'_A) > \frac{k \cdot \text{size}(r[\mathcal{R}_2]) - \text{size}(r[\mathcal{R}_1])}{o_1 - k \cdot o_2} + \text{size}(v_A)$$

This gives us (note that $o_1 - k \cdot o_2 > 0$):

$$(o_1 - k \cdot o_2) \cdot (\text{size}(v'_A) - \text{size}(v_A)) > k \cdot \text{size}(r[\mathcal{R}_2]) - \text{size}(r[\mathcal{R}_1])$$

$$\text{size}(r[\mathcal{R}_1]) + o_1 \cdot (\text{size}(v'_A) - \text{size}(v_A)) > k \cdot \text{size}(r[\mathcal{R}_2]) + k \cdot o_2 \cdot (\text{size}(v'_A) - \text{size}(v_A))$$

$$\text{size}(r'[\mathcal{R}_1]) > k \cdot \text{size}(r'[\mathcal{R}_2])$$

\square

Lemma 5. *Let R be a schema with constraints Σ , and $\mathcal{R}_1, \mathcal{R}_2$ be decompositions of R . If \mathcal{R}_1 does not (c-)dominate \mathcal{R}_2 w.r.t. size, then \mathcal{R}_1 does not (c-)dominate \mathcal{R}_2 w.r.t. attribute count.*

Proof. The proof is analogous to the last one. For every k, r ($k = 1$ for domination) with

$$\text{size}(r[\mathcal{R}_1]) > k \cdot \text{size}(r[\mathcal{R}_2])$$

we need to construct a relation r' such that for some attribute A we get

$$\text{count}_A(r'[\mathcal{R}_1]) > k \cdot \text{count}_A(r'[\mathcal{R}_2]).$$

Again we use that the attribute values occurring in $r[\mathcal{R}_1]$ and $r[\mathcal{R}_2]$ are the same. Since $\text{size}(r[\mathcal{R}_1]) > k \cdot \text{size}(r[\mathcal{R}_2])$, there must exist some attribute value v_A of an attribute A which occurs more than k times as often in $r[\mathcal{R}_1]$ than in $r[\mathcal{R}_2]$. We construct r' from r by selecting exactly those tuples which have the value v_A on attribute A . As Σ contains only functional and join dependencies, these constraints still hold for r' . And clearly we now have $\text{count}_A(r'[\mathcal{R}_1]) > k \cdot \text{count}_A(r'[\mathcal{R}_2])$. \square

We can combine the last two lemmas.

Theorem 6. *Let R be a schema with constraints Σ , and $\mathcal{R}_1, \mathcal{R}_2$ be decompositions of R . Then \mathcal{R}_1 (c -) dominates \mathcal{R}_2 w.r.t. size iff \mathcal{R}_1 (c -) dominates \mathcal{R}_2 w.r.t. attribute count.*

Proof. Follows immediately from the lemmas shown. \square

3.2 Attribute Count vs. Containing Schema Closures

We shall now show that the orders defined by attribute count and containing schema closures are actually the same. One direction of implication is easy to show.

Theorem 7. *Let R be a schema with constraints Σ , and $\mathcal{R}_1, \mathcal{R}_2$ be decompositions of R . If $\mathcal{R}_1 \left\{ \begin{array}{l} \text{weakly} \\ \text{strongly} \end{array} \right\} \text{csc-dominates } \mathcal{R}_2$ then $\mathcal{R}_1 \left\{ \begin{array}{l} c\text{-dominates} \\ \text{dominates} \end{array} \right\} \mathcal{R}_2$.*

Proof. Let r be any relation on R and A some attribute in R .

If \mathcal{R}_1 weakly csc-dominates \mathcal{R}_2 , then for every schema $R_1 \in \mathcal{R}_1$ with $A \in R_1$ there exists a schema $R_2 \in \mathcal{R}_2$ with $A \in R_2$ and $\overline{R_1} \subseteq \overline{R_2}$. By lemma 3 we have $|R_1| \leq |R_2|$, and each such schema R_2 is mapped to at most $|\mathcal{R}_1|$ times. Therefore the number of tuples containing attribute A in $r[\mathcal{R}_1]$ is at most $|\mathcal{R}_1|$ times larger than the number of tuples with A in $r[\mathcal{R}_2]$. Thus \mathcal{R}_1 c -dominates \mathcal{R}_2 with $k = |\mathcal{R}_1|$.

If \mathcal{R}_1 strongly csc-dominates \mathcal{R}_2 , then by lemma 3 and due to the injectivity of the mapping f in definition 9, the number of tuples with attribute A in $r[\mathcal{R}_1]$ is no larger than the number of those in $r[\mathcal{R}_2]$. Thus \mathcal{R}_1 dominates \mathcal{R}_2 . \square

It is possible to show implication in the other direction. This can be done by assuming that \mathcal{R}_1 does not weakly or strongly csc-dominate \mathcal{R}_2 , and constructing example relations which show that \mathcal{R}_1 does not c -dominate or dominate \mathcal{R}_2 . These constructions are extensive though, and we will omit them here. Instead we only present the results.

Theorem 8. *Let R be a schema with functional and join dependencies Σ , and $\mathcal{R}_1, \mathcal{R}_2$ be decompositions of R . If \mathcal{R}_1 does not weakly csc-dominate \mathcal{R}_2 , then \mathcal{R}_1 does not c -dominate \mathcal{R}_2 .*

Theorem 9. *Let R be a schema with functional dependencies Σ , and $\mathcal{R}_1, \mathcal{R}_2$ be decompositions of R . If \mathcal{R}_1 does not strongly csc-dominate \mathcal{R}_2 , then \mathcal{R}_1 does not dominate \mathcal{R}_2 .*

4 Relationship to other Normal Forms

A number of normal forms have been proposed for relational databases, depending on the types of integrity constraints given. For functional dependencies, BCNF and 3NF are the most popular ones. For functional and multivalued dependencies 4NF is the logical extension of BCNF, which for functional and join dependencies has then been extended to PJ/NF, 5NFR and KCNF. The last normal form, KCNF, will be of particular interest to us.

Definition 11. Let R be a schema with functional and join dependencies Σ . Then R is in *Key-Complete Normal Form (KCNF)*, if for every join dependency $\bowtie [R_1, \dots, R_n]$ implied by Σ , the keys among R_1, \dots, R_n cover R . That is, we have

$$\bigcup \{R_i \mid \Sigma \models R_i \rightarrow R\} = R$$

Note that in the case where Σ contains only functional and multivalued dependencies, KCNF is equivalent to 4NF, and when Σ contains only functional dependencies KCNF equates to BCNF [6].

Given a single schema with constraints Σ , the absence of redundancy is precisely characterized by BCNF, 4NF and KCNF. That is, a schema R is free of redundancy iff it is in BCNF, 4NF or KCNF [6].

While our normal form has been designed to minimize size rather than redundancy, the intuition is that minimizing one minimizes the other as well. We will show that this intuition holds insofar, as that when we consider the set of all lossless decompositions, then a single schema is in DNF iff it is in KCNF. Thus DNF can be seen as an extension of BCNF, 4NF and KCNF (when considering all lossless decompositions).

Recall that a decomposition is in DNF if it is minimal among a given set of 'suitable' decompositions, and that for each such set we obtain a different DNF.

Theorem 10. *Let R be a schema with constraints Σ . Then R is in KCNF iff $\{R\}$ is in DNF w.r.t. all lossless decompositions of R .*

Proof. (1) Let R be in KCNF. Let $\mathcal{R} = \{R_1, \dots, R_n\}$ be any lossless decomposition of R . Then Σ implies the join dependency $\bowtie [R_1, \dots, R_n]$, and since R is in KCNF, every attribute $A \in R$ lies in some R_i which forms a key of R . Thus $R \in CSC_A(\mathcal{R})$ for all A , so $\{R\}$ strongly csc-dominates \mathcal{R} . Therefore $\{R\}$ dominates \mathcal{R} by theorem 7. As this holds for all lossless decompositions \mathcal{R} , $\{R\}$ is in DNF.

(2) Let R not be in KCNF. Then Σ implies a join dependency $\bowtie [R_1, \dots, R_n]$ such that

$$\bigcup \{R_i \mid \Sigma \models R_i \rightarrow R\} \neq R$$

The decomposition $\mathcal{R} = \{R_1, \dots, R_n\}$ is lossless, and there exists an attribute $A \in R$ which does not lie in any R_i which forms a key of R . Clearly \mathcal{R} weakly csc-dominates $\{R\}$, and since $R \notin CSC_A(\mathcal{R})$ we have that $\{R\}$ does not weakly csc-dominate \mathcal{R} . Thus \mathcal{R} strictly c -dominates $\{R\}$ by theorems 7 and 8, showing that $\{R\}$ is not in DNF. \square

5 An Example

A university has oral examinations at the end of each semester, and wants to manage related data using a relational database. The relevant attributes to be stored are

$$R = \{\text{Student}, \text{Course}, \text{Chapter}, \text{Time}, \text{Room}\}$$

Here Chapter denotes a chapter from the course textbook the student will be examined about. Every student can get examined about multiple chapters, and chapters may vary for each student. Multiple students can get examined at the same time in the same room, but the course must be the same. Further constraints are that a student gets examined for a course only once, and can't be in multiple rooms at the same time. Those conditions can be expressed through functional dependencies as follows:

$$\Sigma = \left\{ \begin{array}{l} \{Student, Course\} \rightarrow Time, \\ \{Student, Time\} \rightarrow Room, \\ \{Time, Room\} \rightarrow Course \end{array} \right\}$$

We are now presented with the task of decomposing R . If it is deemed necessary to preserve dependencies, a reasonable Boyce-Codd Normal Form decomposition could be synthesized as follows:

$$\mathcal{R}_{DP-BCNF} = \left\{ \begin{array}{l} \{Student, Course, Time\}, \\ \{Student, Time, Room\}, \\ \{Course, Time, Room\}, \\ \{Student, Course, Chapter\} \end{array} \right\}$$

This decomposition, however, is not in dependency preserving Domination Normal Form - it is strictly dominated by the decomposition

$$\mathcal{R}_{DP-DNF} = \left\{ \begin{array}{l} \{Student, Course, Time, Room\}, \\ \{Student, Course, Chapter\} \end{array} \right\}$$

Note that the latter decomposition is in dependency preserving DNF (although we won't show this here), but not in BCNF.

If dependencies need not be preserved, we could use the well-known BCNF decomposition algorithm [3, 4, 5] to obtain the following BCNF decomposition (decomposing first by $\{Student, Course\} \rightarrow Time$, then by $\{Student, Course\} \rightarrow Room$):

$$\mathcal{R}_{BCNF} = \left\{ \begin{array}{l} \{Student, Course, Time\}, \\ \{Student, Course, Room\}, \\ \{Student, Course, Chapter\} \end{array} \right\}$$

Again, this is strictly dominated by the decomposition \mathcal{R}_{DP-DNF} , which is in DNF even among all lossless decompositions. At first look it might seem that \mathcal{R}_{DP-DNF} is strictly c-dominated by

$$\mathcal{R}_{DNF} = \left\{ \begin{array}{l} \{Student, Time, Room\}, \\ \{Course, Time, Room\}, \\ \{Student, Course, Chapter\} \end{array} \right\}$$

A closer look reveals though that both c-dominate each other, since the attribute *Course* already appears in the key schema $\{Student, Course, Chapter\}$ in both cases. The latter decomposition is both in DNF and BCNF. The decomposition

$$\mathcal{R}'_{DP-DNF} = \left\{ \begin{array}{l} \{Student, Course, Time, Room\}, \\ \{Student, Time, Chapter\} \end{array} \right\}$$

however, while in dependency preserving DNF, is not in DNF w.r.t. all lossless decomposition, since it is strictly c-dominated by

$$\mathcal{R}'_{DNF} = \left\{ \begin{array}{l} \{Student, Time, Room\}, \\ \{Course, Time, Room\}, \\ \{Student, Time, Chapter\} \end{array} \right\}$$

Which decomposition to choose is ultimately up to the designer, as storage space and redundancy are

not the only design criteria to consider. The schema $\{Student, Course, Chapter\}$ appears to be a more intuitive choice than $\{Student, Time, Chapter\}$, although deciding this requires domain knowledge which is not encoded in the constraints.

We conclude this example by providing an instance of R and its projection onto the first four decompositions given. While DNF considers all valid instances rather than just a given one, we chose an instance which visualizes the relationship between a schema's closure and the size of relations projected onto it.

<i>Student</i>	<i>Course</i>	<i>Ch.</i>	<i>Time</i>	<i>Ro.</i>
J.C. Denton	Networks	2	3/10, 1pm	101
J.C. Denton	Networks	6	3/10, 1pm	101
J.C. Denton	Security	1	4/10, 1pm	104
J.C. Denton	Security	5	4/10, 1pm	104
L. Nasher	Networks	3	3/10, 1pm	101
L. Nasher	Networks	4	3/10, 1pm	101
L. Nasher	Security	4	4/10, 1pm	104
L. Nasher	Security	7	4/10, 1pm	104
O. Shrek	Networks	2	3/10, 1pm	101
O. Shrek	Networks	8	3/10, 1pm	101
O. Shrek	Security	5	4/10, 1pm	104
O. Shrek	Security	2	4/10, 1pm	104
M. Smith	Security	4	4/10, 2pm	104
M. Smith	Security	6	4/10, 2pm	104
M. Anderson	Networks	3	3/10, 1pm	101
M. Anderson	Networks	5	3/10, 1pm	101
A. Cheng	Networks	2	3/10, 1pm	103
A. Cheng	Networks	4	3/10, 1pm	103
A. Cheng	Security	4	4/10, 2pm	104
A. Cheng	Security	5	4/10, 2pm	104
N. Cheng	Networks	1	3/10, 1pm	103
N. Cheng	Networks	7	3/10, 1pm	103
N. Cheng	Security	5	4/10, 2pm	104
N. Cheng	Security	6	4/10, 2pm	104
J.Zhao	Networks	2	3/10, 1pm	103
J.Zhao	Networks	5	3/10, 1pm	103

All four decompositions share the key schema $\{Student, Course, Chapter\}$, with corresponding projection

<i>Student</i>	<i>Course</i>	<i>Ch.</i>
J.C. Denton	Networks	2
J.C. Denton	Networks	6
J.C. Denton	Security	1
J.C. Denton	Security	5
L. Nasher	Networks	3
L. Nasher	Networks	4
L. Nasher	Security	4
L. Nasher	Security	7
O. Shrek	Networks	2
O. Shrek	Networks	8
O. Shrek	Security	5
O. Shrek	Security	2
M. Smith	Security	4
M. Smith	Security	6
M. Anderson	Networks	3
M. Anderson	Networks	5
A. Cheng	Networks	2
A. Cheng	Networks	4
A. Cheng	Security	4
A. Cheng	Security	5
N. Cheng	Networks	1
N. Cheng	Networks	7
N. Cheng	Security	5
N. Cheng	Security	6
J.Zhao	Networks	2
J.Zhao	Networks	5

The decompositions differ only by the remaining schemas. We obtain the projections:

<i>Student</i>	<i>Course</i>	<i>Time</i>	<i>Ro.</i>
J.C. Denton	Networks	3/10, 1pm	101
J.C. Denton	Security	4/10, 1pm	104
L. Nasher	Networks	3/10, 1pm	101
L. Nasher	Security	4/10, 1pm	104
O. Shrek	Networks	3/10, 1pm	101
O. Shrek	Security	4/10, 1pm	104
M. Smith	Security	4/10, 2pm	104
M. Anderson	Networks	3/10, 1pm	101
A. Cheng	Networks	3/10, 1pm	103
A. Cheng	Security	4/10, 2pm	104
N. Cheng	Networks	3/10, 1pm	103
N. Cheng	Security	4/10, 2pm	104
J.Zhao	Networks	3/10, 1pm	103

<i>Student</i>	<i>Course</i>	<i>Time</i>
J.C. Denton	Networks	3/10, 1pm
J.C. Denton	Security	4/10, 1pm
L. Nasher	Networks	3/10, 1pm
L. Nasher	Security	4/10, 1pm
O. Shrek	Networks	3/10, 1pm
O. Shrek	Security	4/10, 1pm
M. Smith	Security	4/10, 2pm
M. Anderson	Networks	3/10, 1pm
A. Cheng	Networks	3/10, 1pm
A. Cheng	Security	4/10, 2pm
N. Cheng	Networks	3/10, 1pm
N. Cheng	Security	4/10, 2pm
J.Zhao	Networks	3/10, 1pm

<i>Student</i>	<i>Time</i>	<i>Ro.</i>
J.C. Denton	3/10, 1pm	101
J.C. Denton	4/10, 1pm	104
L. Nasher	3/10, 1pm	101
L. Nasher	4/10, 1pm	104
O. Shrek	3/10, 1pm	101
O. Shrek	4/10, 1pm	104
M. Smith	4/10, 2pm	104
M. Anderson	3/10, 1pm	101
A. Cheng	3/10, 1pm	103
A. Cheng	4/10, 2pm	104
N. Cheng	3/10, 1pm	103
N. Cheng	4/10, 2pm	104
J.Zhao	3/10, 1pm	103

<i>Student</i>	<i>Course</i>	<i>Ro.</i>
J.C. Denton	Networks	101
J.C. Denton	Security	104
L. Nasher	Networks	101
L. Nasher	Security	104
O. Shrek	Networks	101
O. Shrek	Security	104
M. Smith	Security	104
M. Anderson	Networks	101
A. Cheng	Networks	103
A. Cheng	Security	104
N. Cheng	Networks	103
N. Cheng	Security	104
J.Zhao	Networks	103

<i>Course</i>	<i>Time</i>	<i>Ro.</i>
Networks	3/10, 1pm	101
Security	4/10, 1pm	104
Security	4/10, 2pm	104
Networks	3/10, 1pm	103

Clearly \mathcal{R}_{DP-DNF} and \mathcal{R}_{DNF} requires less storage space than $\mathcal{R}_{DP-BCNF}$ and \mathcal{R}_{BCNF} .

6 Conclusion

We have introduced a new normal form called DNF for relational databases, based on the type of constraints given and the requirements for a decomposition. Here, a decomposition is in DNF if and only if there is no "better" decomposition among the decompositions considered. The partial orders describing this "better" property have been defined in semantical terms, and for functional and in part for join dependencies, they have been characterized syntactically. Using this syntactical characterization we then showed that, when considering all lossless decompositions, our normal form is an extension to the existing normal forms BCNF, 4NF and KCNF, which have already proven to be useful over the past 30 years.

For multiple schemas DNF appears more suitable than just considering schemas individually, as has been done traditionally. At the same time it is always applicable, in that a decomposition into DNF always exists, even when we restrict ourselves to certain types of decompositions, e.g. dependency preserving ones.

References

- [1] ARENAS, M., AND LIBKIN, L. An information-theoretic approach to normal forms for relational and xml data. In *PODS (2003)*, pp. 15–26.
- [2] BEERI, C., AND BERNSTEIN, P. A. Computational problems related to the design of normal form relational schemas. *ACM Transactions on Database Systems* 4, 1 (1979), 30–59.
- [3] LEVENE, M., AND LOIZOU, G. *A Guided Tour of Relational Databases and Beyond*. Springer, 1999.
- [4] MAIER, D. *The Theory of Relational Databases*. Computer Science Press, 1983.
- [5] MANNILA, H., AND RÄIHÄ, K.-J. *The Design of Relational Databases*. Addison-Wesley, 1987.
- [6] VINCENT, M. W. Redundancy elimination and a new normal form for relational database design. In *Semantics in Databases (1998)*, vol. 1358 of *Lecture Notes in Computer Science*, Springer, pp. 247–264.

JooJ: Real-time Support for Avoiding Cyclic Dependencies

Hayden Melton, Ewan Tempero
 Department of Computer Science
 University of Auckland
 Auckland, New Zealand
 {hayden|ewan}@cs.auckland.ac.nz

Abstract

The design guideline *avoid dependency cycles among modules* was first alluded to by Parnas in 1978. Many tools have since been built to detect cyclic dependencies among a program's organisational units, yet we still see real applications riddled with large dependency cycles. Our solution to this problem is to proactively check for dependency cycles as a developer writes code. In this way a cycle can be identified and eliminated the moment any fragment of code is written that induces one. This approach is analogous to a well-known manufacturing quality assurance technique known as *poka-yoke*. We demonstrate the feasibility of our 'real-time checking' approach via an Eclipse plugin we have built called JooJ.

1 Introduction

Over the years there have been many guidelines proposed for writing effective code. Roughly speaking these guidelines fall into three categories — those pertaining to (1) style, (2) correctness and (3) design. Style guidelines aim to improve the readability of code through consistent naming and formatting (e.g., *Code Conventions for the Java Programming Language* (Sun 1999)). Correctness guidelines aim to help programmers avoid common or subtle errors (e.g., "Class overrides equals() without overriding hashCode()" (Bloch 2001)). Design guidelines aim to help programmers make decisions about the internal structure of a system (e.g., Riel's *Object-Oriented Design Heuristics* (Riel 1996) and Design Patterns (Gamma, Helm, Johnson & Vlissides 1995)).

There are many tools currently available for checking conformance of Java code to style, correctness and design guidelines. We are interested in those that provide continuous (or proactive) checking as opposed to those that are run intermittently, at a developer's discretion. The Eclipse Integrated Development Environment (IDE) is a good example of a tool that proactively checks Java code against style and correctness guidelines. As the developer enters code into Eclipse it is analysed in 'real-time' for problems (e.g., syntax error, unused local variable, unparameterised use of a generic type etc). In this way the developer gets immediate feedback about some aspects of the quality of his code. The importance of this immediacy is evident in a well-known aphorism: that it's cheaper to fix problems earlier in the development process than later (Pressman 2001, p.197-198).

While 'real-time' code analysis has successfully been implemented by Eclipse (and other IDEs) for supporting correctness and style guidelines it seems that there are few, if any, tools available that take this approach to supporting *design guidelines* at the level of source code. We believe one reason for this is that often it is computationally more expensive to analyse code for design guidelines than to do

so for style and correctness guidelines. This is because many design guidelines, especially the one in which we are interested in, provide advice about structuring of the whole system and cannot be determined solely through the analysis of a single source file.

Another reason why design guidelines may not be supported through real-time code analysis is that it is often difficult to determine a satisfactory measure for a design guideline from source code. In the case of module cohesion, for instance, there have been numerous metrics presented that can be automatically computed from source code (see (Briand, Daly & Wust 1998) for example) yet none is widely accepted or even used by practitioners. Fortunately the design guideline in which we are interested does not suffer from this measurement problem.

In this paper we present a tool we have developed to determine the feasibility of proactively supporting the design principle *avoid dependency cycles among modules* through real-time source code analysis. Our tool, JooJ (pronounced "Joo-jay"), has been developed as a plugin for Eclipse. It transparently extends the style and and correctness checking already provided by Eclipse.

The remainder of the paper is organised as follows. In Section 2 we review the design principle JooJ supports and discuss the motivation for JooJ. In Section 3 we give an overview of JooJ's expected user interface and features. In Section 4 we discuss some of the details of JooJ's implementation. In Section 5 we evaluate the performance of JooJ in terms of time and space. In Section 6 we review other cycle-detecting and real-time analysis tools. Finally, in Section 7, we draw conclusions from this work.

2 Background and Motivation

Software *design guidelines* guide the decisions developers make about the internal structure of a system. They help us to structure a system in a way that makes it easy to understand, test, modify, reuse and so on. The design guideline relevant to this paper is *avoid dependency cycles among modules*. Dependencies among the source files of an application are a natural consequence of modularisation. In dividing a program up into modules we break it up into more manageable parts, but these parts must collaborate in order to provide the functionality of the system as a whole. It is these collaborations that cause dependencies.

2.1 Impact of Cycles

Parnas was the first to discuss the effect dependency cycles among a program's modules might have on software quality attributes (Parnas 1978). He argued that when two modules were cyclically dependent neither could not be tested, build or reused independently of the other. When there are long dependency cycles encompassing many modules Parnas argued that we might end up with a system where no single part of the works until all the rest of it works.

The most comprehensive work on cycles in the context of the Object-Oriented (OO) paradigm is by Lakos. He states that cycles among the source files of C++ programs inhibit understanding, testing and reuse (Lakos 1996, p.185), and that cycles among packages inhibit development, marketing, usability, production and reliability (Lakos 1996, p.494-495).

Other design guidelines also support the “avoid cycles” guideline. For instance, Riel states “Derived classes must have knowledge of their base class by definition, but base classes should not know anything about their derived classes” (Riel 1996, p.81). Disallowing the dependency of a base classes on its derived classes prevents a dependency cycle between the base and derived classes. Stevens et al. state the design guideline *minimise coupling between modules*. A design with dependency cycles has higher coupling than its acyclic analog (e.g., if modules A and B are in a cycle then B has higher coupling than if the only dependency is from A on B). Booch says “...all well structured object-oriented architectures have clearly defined layers” (Booch 1995). Long dependency cycles make it difficult to divide a system’s classes into clearly defined layers, where classes in a given layer can only depend on others in lower layers.

If a cyclic dependency exists, then the question arises as to how to remove it. This must involve removing a dependency, and so breaking a collaboration, but which one? Lakos provides some advice on deciding which dependency to break but this advice often relies on characteristics of the problem domain (e.g., this object is more “primitive” than that). Many OO design guidelines also provide advice. For example, if one class is “a part of” another, then the other must always depend on it, whereas any dependency by the part on the whole is not always necessary. Similarly, a subclass must always depend on its parent, but a parent should not depend on any of its children. In a model-view-controller design, the view must depend on the model, but *the model of an application should not depend on its view* (Riel 1996, p.36). What this means is that it does not make sense to remove some dependencies, so we must provide some means to manage such dependencies, a point we return to in Section 3.

2.2 Definition of Cycle

We have adapted Lakos’ work with cycles in C++ to Java (Melton & Tempero 2006). For simplicity of explanation, we assume all “top-level” classes are declared in separate .java source files. This means that for a class A, A.java and A.class refer to the same entity, and we will use these interchangeably. There are several subtle variations on the definition of “dependency”, particularly with regard to differences between Java 5 and its predecessors. These variations are discussed in our adaptation of Lakos’ work (Melton & Tempero 2006). Our tool can cope with each of these, and it is sufficient for our presentation to use the simplest: A class A DependsOn a class B if it needs B.class on the classpath in order to compile.

2.3 Prevalence of Cycles

Our main motivation for providing tool support to avoid dependency cycles comes from an empirical study we performed (Melton & Tempero 2006). The results of this study indicate that not only do cycles exist in many Java applications, but they are often large and complex. In our study we analysed a corpus of 78 real, open- and closed-source Java applications and found that:

- Two commercial applications each had a single long cycles involving over 2000 top-level Java classes.
- Eight out of the 78 applications had single long cycle involving over 500 classes.

- Two popular, widely-downloaded, open source projects (Azureus and Hibernate) had more than half their classes involved in one big cycle.
- Close to 40% of the applications in the corpus had a single cycle involving more than 100 classes.

These results astonished us. They support a claim made by Foote et al. that the most frequently deployed software architecture is the *Big Ball of Mud* (Foote & Yoder 2000). They also justify the large amount of research that has been done on stubbing to break dependency cycles for integrating testing (Hashim, Schmidt & Ramakrishnan 2005, Briand, Labiche & Wang 2003) (Binder 1999, p.980-985). More importantly these results *strongly* motivate the need for a tool to help prevent cycles ever appearing in source code. If we could prevent cycles appearing in a system’s source code, as advocated by Lakos (Lakos 1996) and others (Binder 1999, p.984), then there would be no need for stubbing — an activity Binder identifies as potentially risky, expensive, difficult, and inadequate in the presence of large complex cycles (Binder 1999, p.983-984).

2.4 The Need for Real-time Feedback

There are already many tools for supporting *avoid dependency cycles* in Java (e.g. ByeCycle, Classycle, Dependency Finder, PASTA tool, JDepend, Lattix LDM, see Section 6). The majority of these tools take a batch-style type approach to supporting these principles. The prevalence of dependency cycles in real-world Java software indicates that either these tools are ineffective or software developers do not care much about avoiding dependency cycles. The sheer number of these (mostly free) tools makes it difficult to believe the other alternative: that developers ‘just don’t know’ about their existence.

The problem with batch-style tools is that they do not allow problems to be fixed at the same time they are created. Two important reasons why cycle-causing code is hard to change retrospectively are:

Code is more resistive to change after it has been written. Imagine we are oblivious to a cyclic dependency induced by a statement we have just written. Without tool support this is likely because there is no way to tell if a statement induces a cyclic dependency simply by looking at a single source file — yet this is the way we edit and view source files, one at a time. We then write more statements that depend on the initial cycle-inducing one (and possibly inducing more cycles themselves). Eventually we get around to running our cycle detecting tool and discover the cycle. We are now faced with the task of figuring out how to change or move that statement (and its dependent statements) to break the cycle, and all the while not inducing new, different cycles.

The alternative is that we are informed as soon as we write a statement inducing a cycle. Instead of continuing we can remove the cycle at that point in time (for example by *escalating* (Lakos 1996, p.215-228) that statement to a new or existing higher-level class). The effort involved in making changes to remove the cycle is now limited to dealing with just one statement.

Changing other people’s code is hard. Imagine that another developer wrote the cycle inducing statements, but neglected to run or take notice of the output from our batch-style cycle tool. Now we have to change his code. We may introduce a bug in doing so if we fail to understand all the pre- and post-conditions of his code. We have to spend comparatively more time working out what someone else’s code does. If we cannot understand the code or feel the risk regression from improving its structure is too high we may leave the code as it is. Over time the cycle may grow and grow until it encompasses most of the classes in the system, then the system will have to be thrown away and rewritten from scratch. Indeed cy-

cle growth and throwing systems away are phenomena we have reported (Melton & Tempero 2006).

Consider now the possibility that developers ‘just don’t care’ about avoiding dependency cycles, or that it is a very low priority. As Foote et al. state “[software] architecture frequently takes a back seat to more mundane concerns such as cost, time-to-market, and programmer skill” (Foote & Yoder 2000). We argue real-time, integrated tool support for avoiding dependency cycles can make developers care, and help ensure design principles do not take a back seat to more ‘mundane’ concerns.

Before we (the authors) started using Eclipse (3.1.1) we were unaware of variable declarations in a class that were unused, or variables whose values were assigned by never read from, or unused private methods. Now when we write code, we are immediately informed by Eclipse of these problems (and others) through yellow squiggly underlines of individual statements. Slowly but surely we started taking heed of this feedback as we coded. Continuous ‘micro-refactorings’ to eliminate these problems are now part of our personal coding styles. We suspect that there is a psychological force that drives us to fix statement with yellow squiggly lines under them. We must, of course, fix statements with red squiggles beneath them because these are compilation errors. (We note that in the mid-90’s Microsoft Word was changed to include continuous checking of spelling and grammar and that again, with this feature, we are compelled to get rid of the squiggles as soon as they appear).

2.5 Wider Perspectives

The notion of preventing problems before they occur, or early in the production process, has been around for a long time in the manufacturing industry. In the 1960s an engineer at Toyota called Shigeo Shingo used the term *poka-yoke*, which means ‘mistake-proofing’, to describe this approach to quality assurance. A poka-yoke device aims to prevent potential quality problems before they occur or rapidly detects them as they are introduced (Pressman 2001, p.214-215).

Pressman (Pressman 2001, p.215) states that an effective poka-yoke device exhibits the following characteristics: (1) it is simple and cheap, (2) it is part of the process and (3) it is located near the process task where the mistakes occur. Indeed it can be argued that Eclipse’s style and correctness guideline checking is an effective poka-yoke device because it it brings checking closer to the activity of typing out code than batch-style tools. It also rapidly detects problems as they are created. The effectiveness of our tool, JooJ, can be argued in a similar fashion.

2.6 Applicability

It has been claimed that avoiding dependency cycles among modules is most applicable to large-scale software systems (Martin 1996, Lakos 1996). Martin define large in the context of C++ as 50,000 LOC or more (Martin 1996) and Lakos defines large as in the same context as having “hundreds of header files” (Lakos 1996, p.11). The question we try to address here is *to what proportion of the world’s Java software is our tool applicable?*

A distribution of size in terms of number of classes in the Java corpus of a previous work (Melton & Tempero 2006) is shown in Figure 1. The x-axis represents the number of .java files in a system and the y-axis represents the proportion of applications in the corpus that comprise *at least* that many .java files. So from this chart we can see that about 30% of the applications in the corpus comprise at least 1000 .java files. About 15% comprise at least 2000 .java files. If the corpus used to generate this plot is representative sample of real-world Java software, and we define large as 1000 .java files,

then the support provided by JooJ is applicable to around 30% of the world’s Java software.

If we do not consider the corpus to be a representative sample of real-world Java software then consider what Fayad et al. have to say:

While a 100,000 source line program was a significant undertaking 20 years ago, the typical shrinkwrapped software product today embodies at least that many lines of code. While it is extremely difficult to identify a cost figure, it appears that smaller groups are developing larger programs. This suggests that smaller groups need some of the software methodologies developed for large-scale projects ... (Fayad, Laitinen & Ward 2000).

The implication of this is that large-scale software design guidelines are becoming more and more relevant, as even small companies are capable of building large-scale software systems.

One final statement from Booch implies we should consider applying large-scale software design principles even to small software systems, because it is these systems that often grow into larger, unwieldy ones:

...I see in Java a phenomenon I’ve seen too many times before: simple systems that work well have a nasty way of evolving into big systems that sputter and breakdown and collapse of their own sheer weight. Furthermore, try to scale development techniques that work well for simple systems and you’ll fail: the sustainable development of large complex systems requires fundamentally different techniques than heroic programming efforts offer (Booch 1996, p.208).

Lakos expresses a similar view (Lakos 1996, p.xxvi) and indeed this is our view. We even have empirical evidence to support the notion that small systems often grow into large ones (Melton & Tempero 2006). The argument then is that design guidelines aimed at large-scale software systems should also be applied to small systems. The point of JooJ is to reduce the burden of applying *avoid dependency cycles* to Java code.

3 JooJ

JooJ is a tool to support the design guideline *avoid dependency cycles*: (1) for new and existing Java code; (2) in real-time; (3) in an integrated fashion.

By ‘new and existing Java code’ we mean it supports code that is being written for a new system and code that is being written to maintain (e.g., extend or fix bugs) an existing system. We overload this phrase by also defining it to mean Java 5 (new) and Java 1.4 and earlier (existing). As we will see shortly there are different challenges in supporting the design principles for different versions of Java; and for new and existing systems.

By ‘in real-time’ we mean that Java code is analysed for the design guideline as it is being written. By ‘in an integrated fashion’ we mean that JooJ is an Eclipse plug-in that transparently extends the style and correctness checking that is already built in to Eclipse 3.1.1.

3.1 User Interface

JooJ’s user interface (UI) is no different from that of Eclipse’s built-in style and correctness checking. This means that using JooJ is non-invasive because Eclipse users are already familiar with its UI metaphor. We review the user interface of style and correctness checking in Eclipse order to put JooJ’s UI in context.

Figure 2 is a screen dump from Eclipse’s Java editor. Besides the syntax highlighting it has several ‘annotations’ that are not available in standard text editors. The

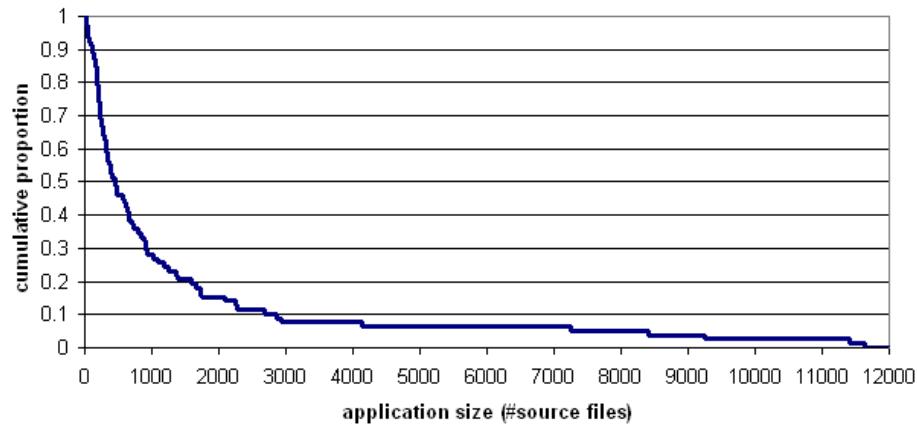


Figure 1: Distribution of application size across 78 Java applications

```

6 public class MyClass {
7
8     private List list = new LinkedList();
9
10    private Object obj;
11
12    public void meth() {
13        String s = "myString";
14        list.add(s);
15        list.foo();

```

Figure 2: Style and correctness checking in Eclipse

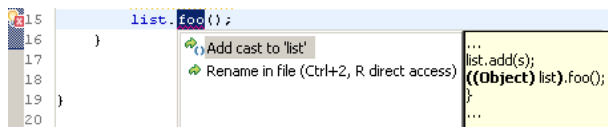


Figure 3: Refactoring suggestions for style and correctness violations in Eclipse

first of these annotations are the *squiggles*¹ on lines 10, 14 and 15. These squiggles indicate that there are problems with the code. The red squiggly on line 15 indicates a compilation error — *the method 'foo' is undefined for type List*. The yellow squiggles on lines 10 and 14 respectively indicate that *references to the generic type List<E> should be parameterised* and that *the field 'obj' is never read locally*. Although not shown in Figure 2 a description of the problem that leads to each squiggly appears as a tooltip when the mouse is hovered over it. Also not evident in Figure 2, but of particular importance is that the squiggles are continuously recomputed as text is typed into the Java editor.

Another annotation evident in Figure 2 is the appearance of lightbulb icons in the left margin on the lines where squiggles occur. Clicking on the lightbulb of line 15 causes a popup to appear as shown in Figure 3. This popup is referred to in the Eclipse documentation as *code assist* or *content assist*. The code assist in Figure 3 presents a list of refactorings that can be performed to correct the problem. In the case of line 15 the code assist is suggesting casting the variable reference `list` to a subtype in the hope that a subtype of `List`'s declared type declares a method `foo()`. The yellow tooltip to the right of the code assist shows the text that will result as a consequence of performing the selected refactoring.

The user interface we are building for JooJ is no different to that illustrated above. If a statement causes a cyclic dependencies then it gets a squiggly under it. If the de-

pendency is in a Strongly Connected Component (SCC) (of size >1) then it gets an orange squiggly beneath it. If the dependency is in the Edge Feedback Set (EFS) computed by JooJ then it gets a magenta squiggly beneath it. Both SCC and EFS are discussed below.

In terms of the lightbulb annotations that provide specific code transformations to fix problems we are also currently in the process of extending JooJ to support the specific refactorings proposed by Lakos (e.g., escalation, demotion, dumb data, manager class etc) (Lakos 1996, ch.5) for breaking cycles. This is actually a difficult problem because as we noted in Section 2 it is often the case that a cycle inducing statement has many dependent statements in the context of its source file. In order to remove the cycle inducing statement we must also move its dependent statements.

3.2 SCC and EFS

A subgraph S of another (directed) graph G is a Strongly Connected Component (SCC) if all of the vertices in S are mutually reachable in G and no additional vertices can be added from G to S that meet this criterion. A vertex is considered reachable from itself. In the context of our problem the vertices of S are classes that are all cyclically dependent, and indeed this is why it is SCCs in which we are interested.

A Minimum-Edge Feedback Set (MEFS) is the smallest set of edges that when removed from a (directed) graph G cause it to become a Directed Acyclic Graph (DAG). Equivalently it makes G a graph with SCCs all of size 1. In the context of our problem the MEFS represents the smallest set of dependencies that when removed break all cycles.

In JooJ the SCCs are computed using an linear-time algorithm presented Cormen et al. (Cormen, Leiserson & Rivest 1990, p.489). Its cost (and implementation) is roughly equivalent to two depth-first searches. Computation of a 'small' edge feedback set is done in JooJ using linear-time a heuristic proposed by Eades et al (Eades, Lin & Smyth 1993). We call the output of Eade's algorithm a *mEFS* to distinguish it from a *MEFS* — the computation of which is NP-complete (Skiena 1998).

3.3 Dependency Removal

There are different challenges for eliminating dependency cycles from newly written code and code that is part of an existing system. If a system is built from scratch using JooJ as a design critic then it is likely that every cycle that appears in the system can be eliminated by the developer the instant it appears.

In existing systems however there are often many classes in large SCCs (Melton & Tempero 2006). As dis-

¹This is the term used for these wavy, coloured underlines in the Eclipse help documents

cussed in Section 2, there are domain-dependent dependencies that should never be removed. JooJ maintains an “exclusion set” of dependencies specified by the user that are never included in the edge feedback set it computes for each SCC.

To support specification of the exclusion set, and to generally support the user understanding the structure of the dependencies, JooJ provides a visualisation of the source types on which a class depends using JUNG². In the visualisation, types are depicted as vertices (labeled with their fully qualified names) and edges represent dependencies. Edges are coloured differently depending on their membership — if an edge is in the edge feedback set it is magenta, if any other edge participating in a SCC it is orange, and other edges are black. A user of JooJ can add to the exclusion set by selecting edges in the visualisation.

4 High-Level Operation

JooJ is able to detect cycles among the classes defined in an application’s .java files. It does not need to deal with classes defined in external libraries (such as the API) because if these libraries are truly external their classes cannot have any compilation dependencies on the application’s classes. Also, as in previous work (Melton & Tempero 2006), JooJ only considers dependencies within the body of a class; and the dependencies of nested classes and inner classes are merged with their top-level counterparts. In this way redundant import statements causing dependency cycles are ignored. This is desirable because the dependencies caused by redundant import statements are superficial; and Eclipse already has a feature to eliminate these redundant imports.

JooJ models a project’s dependencies with the following data structures:

- A map from an Eclipse resource identifier (which is stable across Eclipse sessions) for a compilation unit to the top-level classes that this compilation unit defines. Call this map R , as in resource.
- A map from fully qualified top-level class names to the fully qualified names of that class’s direct supertypes. Call this map S , as in supers.
- A map from fully qualified top-level class names to the fully qualified names of the classes it directly depends on. Call this map D , as in depends on.
- A map from resource identifier for a compilation unit to the latest filesystem timestamp for its corresponding file. Call this map T , as in timestamp.
- A list of SCCs.
- The mEFS for the current SCC.

During an Eclipse session a project’s .java files are opened in the Java editor, examined and modified. As these events occur Eclipse notifies JooJ and it updates its internal data structures to keep the dependency data structures consistent with the changing .java files. The events and updates they cause are described below.

Startup. When JooJ is attached to a particular project it first determines if it has been attached to that project before. If this is the first time the project has been seen by JooJ then all of the .java files in the project are turned into ASTs one-by-one and the dependency data structures are populated for the first time. This can take several minutes, and is discussed further in Section 5.

If JooJ has processed the project before then the dependency data structures are loaded from text files stored in the project’s directory (see the shutdown event). Sometimes a .java file has been changed outside Eclipse, between Eclipse sessions. JooJ detects this situation by comparing the filesystem timestamp of each .java to that

in R . Changed files have to have their dependencies recomputed as if they were modified in an Eclipse session. The types of changes that can happen to a file discussed shortly.

File Contents Changed. If a file has been modified then JooJ leverages Eclipse’s Java Development Tools (JDT) API to turn a .java file into an Abstract Syntax Tree (AST). It then visits the AST in order to determine the modified .java file’s new dependencies (i.e., its top-level type, its super types and the other source classes it *DependsOn*). There are several different types of changes to a .java file and the way they affect the dependency data structures are explained below:

- **Dependencies for compilation unit unchanged.** If a file is changed it is possible that no new type was added to it, and that no types were added or removed from usage in it. We can easily determine this by comparing the supertypes and dependencies of the changed class, to that stored in S and D respectively. If they are unchanged we need not take any further action except to update the positions of the squiggles in the user interface.
- **Dependencies for compilation unit added.** If a dependency is added then we need to update one or more of the maps. If the dependency is added as a supertype we need to update S and D . If the dependency is added in the body of the class then we need to update just D . We also need to update the SCC set if the dependency is not already in the class’s SCC. We need to recompute the class’s SCC’s mEFS.
- **Dependencies for compilation unit removed.** Update S and D and recompute SCC and mEFS.
- **Fully qualified name of top-level type changed.** This situation occurs when the class’s package is changed, or the top-level type is renamed. In this situation the .java file containing the class will eventually have to be renamed or moved directories in order for it to compile. Eclipse models the movement/renaming of files as a remove and then add event. Thus we discuss this situation under the guise of these events.

File Added. Sometimes a new source file is added to a system. In most cases this does not affect the bindings existing files. However if we refer to a type in a source file before we create that type then adding a new .java file (and its type) can affect the dependencies of other files. So when an new type is added we leverage Eclipse’s Java Search facility to find existing references to this type and update the R , D and S correspondingly. After this we compute the SCC and mEFS for the newly added type.

File Removed. Sometimes a source file is removed from the system. Usually this means that a type is removed from the system, unless the same type is declared in two different source files. So we examine R to ensure the type has been removed from the system (i.e., it isn’t declared in other .java files). If it has been removed we update R , S and D to remove all references to the removed type.

File Renamed/Moved. As previously stated Eclipse models this as a removal of a file and the addition of a new one. These are the canonical events that JooJ receives from Eclipse so the updates to the dependency data structures for this situation have already been discussed.

Shutdown. JooJ writes all the dependency maps to the project directory on disk. This saves time during the next startup because the dependencies for each .java file do not have to be recomputed from scratch.

5 Evaluation

5.1 Performance

Much of the design of Eclipse has been influenced by a desire to make it scalable so users can leverage it to

²<http://jung.sourceforge.net/>

develop even large projects comprising thousands source files (Arthorne & Laffra 2004, p.338). Scalability is of particular importance to JooJ because the design principle it supports is primarily for large scale systems. In this section we evaluate the runtime performance of the algorithms implemented by JooJ on 12 open source projects ranging in size from 48 to 11,413 .java files. All of these benchmarks were done on a machine with fairly modest ‘specs’ — an Intel P4 3.2 GHz with 1GB of RAM running Windows XP SP2.

We computed these benchmarks by writing a small program to load the dependency text files stored by JooJ in each project’s directory into memory. We were then able to run the algorithms on the data structures populated with the information from these text files. The data structures used were identical to those implemented in JooJ. The recorded running time of the algorithms does not include the time taken to load the text files.

5.1.1 Algorithms

The time taken in milliseconds to compute all the SCCs from the internal data structures used by JooJ is shown in the ‘SCC’ column of Table 1. This was computed by timing 100 consecutive runs of the algorithm and taking the average. Recalling the SCC algorithm previously described we can infer that the cost of this algorithm is about the same as the cost of two DFSs.

The time taken in milliseconds to compute the mEFS for all the SCCs in each applications dependency graph is shown in the ‘mEFS’ column of Table 1. Again this was computed by timing 100 consecutive runs of the algorithm and taking the average. Recall that our implementation of this algorithm takes SCCs as input. We do not include the time taken to compute these SCCs in this measurement since this is already shown in the ‘SCC’ column.

So from the results in the ‘SCC’ and ‘mEFS’ columns of Table 1 we can infer that the absolute worst case for computing a class’s SCC and the mEFS for that SCC is the sum of these two values. For Eclipse, we could (in the worst case) expect close to a 900ms delay after we change a file and its dependencies have been computed before we can update the statements in the Java editor with squiggles if they are causing cycles. We think that even this worst case delay is acceptable because writing code is inherently slow — we find we spend a lot of time staring at the screen thinking compared with actual typing.

But the worst case is not the typical case. As we described in Section 4 we do not have to recompute a class’s SCC and its mEFS after every change to that class. Many times a dependency added to a class is already in the SCC so we can skip computing this and only have to compute the mEFS. Furthermore, we do not have to compute the mEFS for all SCCs, like we did for the benchmark. We only have to compute the mEFS for the SCC the class is involved in. If the SCC is small (e.g., 50 classes) then the mEFS algorithm takes only a few milliseconds, as if it were computing all the SCCs for a small application like junit, jgraph, jedit or jung.

5.1.2 Data Structures

JooJ maintains a ‘master list’ of strings representing the top-level types declared in the application. When the dependency data structures are populated the strings are drawn from this ‘master list’ so we can have equality-by-reference semantics for our DFS algorithm; and so we can reduce the amount of memory JooJ requires for each project. Table 1 shows the space requirements in number of characters for each of the applications. This was computed by concatenating all the strings in the ‘master list’ for each project and calling `length()` on it.

The size of Eclipse 3.1’s ‘master list’ is about 600,000 characters (as shown in Table 1). If we remove this ‘mas-

ter list’ and allow different instances of the lexically equal string then we have found that the space required for the strings in JooJ’s dependency data structure for Eclipse can grow to about 6,000,000 characters. So maintaining a ‘master list’ can reduce the space demands of JooJ (at least in terms of strings) by a factor of 10. In fact, by maintaining a master list of strings it may be the overhead of the data structures (i.e., the HashMaps and LinkedLists that dominate JooJ’s space requirements for a project.

5.1.3 Eclipse API

The SCC and mEFS algorithms are really only half the story when it comes to performance. These algorithms operate on adjacency list representations of class dependency graphs. The actual dependencies must be computed from the text of .java files. In order to do this we leverage Eclipse’s JDT. We use the JDT to create ASTs and use *bindings* in order to resolve a name to the type to which it refers. It is well-documented in the Eclipse API that bindings are expensive (time and space-wise) to create. But we must recompute the bindings for a .java file each time it is changed so we need to know how long this takes.

Figure 4 shows the time taken to create an AST from a .java file using Eclipse’s `ASTParser` class. The files were chosen at random from Ant — we couldn’t easily select a hodgepodge of files from different applications because a source file requires the context of its application in order to compile (and compute bindings). The x-axis of the graph represents the size of the class in non-comment source statements (found by counting ‘;’ and ‘{’ characters not in comments). The y-axis represents the time taken (ms) to construct an `ASTParser` instance, create an AST, and visit the ASTs bindings to determine its dependencies.

There are 3 series on the graph that correspond to three options in using `ASTParser`. The first series ‘no bindings’ shows the amount of time taken to create an AST without bindings. This is a baseline so we can see how much bindings actually cost. The next series shows what we term ‘single bindings’ because it uses the `ASTParser` in a way appropriate only for a single compilation unit (using the `setSource` and `getAST` methods). The next series ‘batch bindings’ shows the performance of the `ASTParser` when it is to parse just a single file in ‘batch’ mode (by calling the `createASTs` method). Interestingly using batch mode for a single file appears to be much slower than using it for a single compilation unit. This was not stated in the API, and indeed we only discovered the single compilation unit mode late in the development of JooJ.

Figure 5 is another view of the data in Figure 4. In this plot the x-axis represents time (ms) to create the AST under each of the conditions. The y-axis represents the proportion of .java files from our sample that will have parsed within the given time. So we can see from this plot that using single bindings about 80% of source files will have parsed within 100ms. Almost 100% of source files will have parsed within 200ms.

There are some issues in collecting the data of Figures 5 and 4 that necessitate further discussion. Firstly Eclipse maintains a Least Recently Used (LRU) cache of a project’s resources (Arthorne & Laffra 2004, p.338). In order to ensure we were not measuring the time to load a resource from disk into memory we creating consecutively created 10 ASTs for each of the files but only measured the time taken to process the last 9. In this way we could be fairly sure that the .java file’s contents was cached in Eclipse for our measurements. This is a reasonable thing to do because JooJ operates on the file a programmer is editing, which necessarily must be already in memory.

Finally, when JooJ is first attached to a project it must compute the bindings (dependencies) for all of the .java files in that project. We determined that doing this for Ant

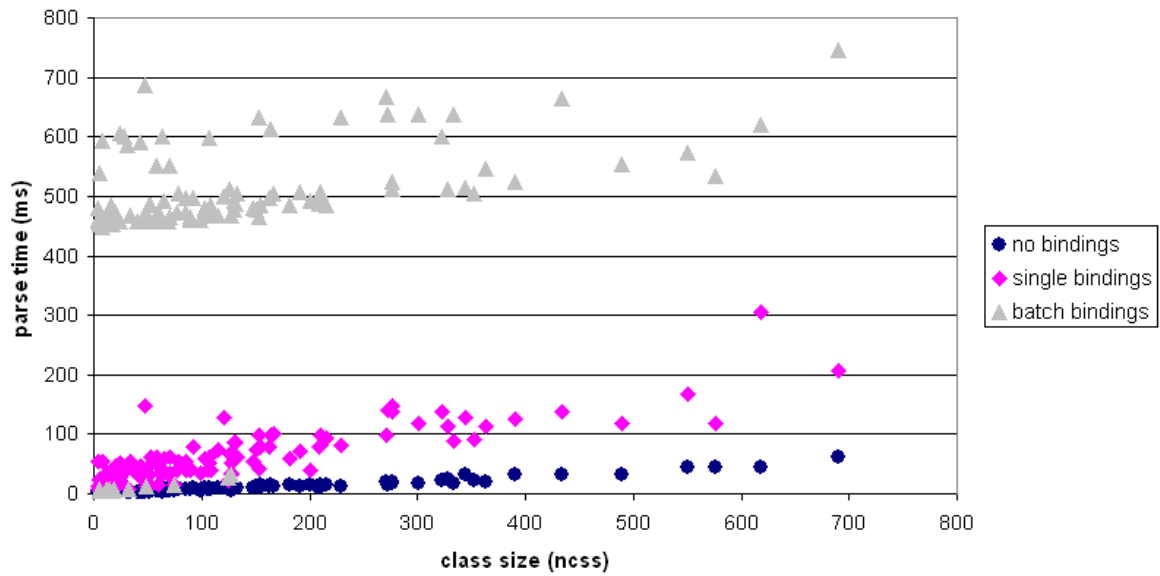


Figure 4: Time to create ASTs for a sample of . java files

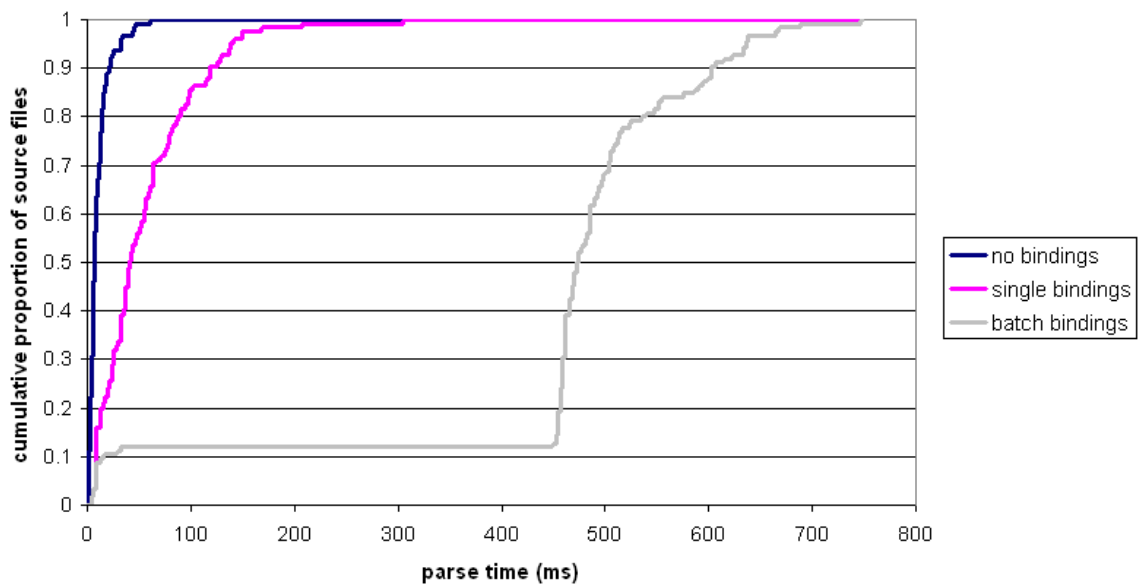


Figure 5: Distribution of AST creation times with and without bindings

<i>Application</i>	<i>Size (classes)</i>	<i>SCC (ms)</i>	<i>mEFS (ms)</i>	<i>Space (chars)</i>
junit-3.8.1	48	0.3	0.2	1325
jgraph-5.7.4.3	50	0.6	1	1559
jedit-4.2	234	2	7	8437
jhotdraw-6.0.1	300	3	1	11501
jung-1.7.1	454	5	0.7	22893
ant-1.6.5	700	7	6	34160
tomcat-5.0.28	892	10	4	36494
hibernate-3.1-rc2	902	12	98	34884
argouml-0.18.1	1251	18	63	61936
azureus-2.3.0.4	1650	25	174	91547
netbeans-4.1	8406	281	80	445691
eclipse-SDK-3.1	11413	506	424	616328

Table 1: Algorithm performance

takes close to 25 seconds. This equates to an average of 36ms per file. The next time we attached JooJ to Ant it took less than 1s to load the text files on disk into memory and compare the time stamps of the loaded .java files to those JooJ wrote to disk when our Eclipse session was last terminated.

6 Related Work

6.1 ByeCycle

ByeCycle³ is a tool that is very similar to JooJ in that it checks for cycles among classes in ‘real-time’. However the primary feature of ByeCycle is a visualisation of the cycles a class is involved in. Also the granularity of its updates appears to be limited to when a file is saved or loaded, not as code is keyed in.

We have used ByeCycle and found JooJ offers several advantages over it. JooJ allows the dependencies that create a cycle to be related back to their corresponding statements in the source code. JooJ also determines all cyclic dependencies among classes whereas ByeCycle condenses classes outside the current class’ package into packages. In this way it appears that only the packages on which the class directly depends are analysed meaning ByeCycle does not perform whole program analysis. Presumably this is due to the screen real estate available for visualisation of cycles.

Also JooJ computes a mEFS and uses this to aid a developer decision where in the source code to break a cycle. Finally JooJ computes both definitions of *DependsOn* (Melton & Tempero 2006) (one for Java 1.4 and below, and one for Java 5) meaning it allows ‘necessary cycles’ (i.e. those expressing *intrinsic interdependency*) to be expressed in a type-safe fashion.

6.2 Design Level Tools

There are several tools available that do ‘real-time’ checking of a UML design. ArgoUML (Robbins, Hilbert & Redmiles 1998) may well have been the first of these tools. It provides several types of design critics pertaining to correctness, completeness, optimisation, alternatives, evolvability, presentation, experience and organisation that are continually evaluated against a design. A ‘todo’ list continuously updated by these critics with suggestions for the improvement of a design.

Egyed (Egyed 2006) has also produced a tool *UMLAnalyser* that checks the consistency of UML diagrams against one-another in ‘real-time’. One of the motivators for his tool is that apparently the consistency critics in ArgoUML are not able to keep up with an engineer’s changes to a large UML model. Both Egyed’s tool and ArgoUML differ from JooJ in that that operate at the design phase, rather than the coding (or implementation) phase.

³<http://byecycle.sourceforge.net/>

6.3 Batch-style Cycle Tools

There are a plethora of other batch-style cycle checking tools for Java. Classycle⁴ searches for cyclic dependencies among the classes of a Java application by analysing byte-code. This is problematic because the system has to be in a compilable state for it run. JooJ does not require this because Eclipse’s bindings work even in the presence of many forms of compilation errors.

JDepend⁵ analyses .class files in order to find cycles among packages. Again this tool operates on byte-code files. Also it does not detect SCCs, only cycles found during a DFS: “cyclic dependency detection may not report all cycles reachable from a given package. The detection algorithm stops once any given cycle is detected”. Hautus’s Package Structure Analysis (PASTA) tool (Hautus 2002) is also geared towards finding cycles among packages. It provides a visualisation of the package structure and tries to, much like JooJ, identify the smallest set of dependencies required to break all cycles among packages. Again it should be noted there cycles among packages do not necessarily imply cycles among classes so these tools solve slight different problems. In a sense finding cycles among classes is a more fundamental problem because if there are large SCCs of classes then there cannot be an acyclic package structure (Melton & Tempero 2007).

Lattix LDM (Sangal, Jordan, Sinha & Jackson 2005) is another Eclipse plugin we have discovered similar to JooJ. It allows detection of cycles and allows specification of the ‘dominance’ relation among packages. It differs from JooJ in that abstracts away from the actual source code with a table known as a Dependency Structure Matrix (DSM). Allowable and undesirable dependencies are shown in this matrix at apparently at the granularity of the package rather than class. It also appears that this tool does not do real-time checking (a press release states it can be “automatically synchronized with every build”) and the UI appears to take over from the Java editor where code is typed. We think JooJ is a more effective poka-yoke device on the basis that it detects problems more quickly and at the activity that creates them (coding, not doing a software build).

7 Conclusions

We believe there should be real-time support for design guidelines that apply to the *whole program*. We have demonstrated the feasibility of doing so for the *avoid dependency cycles* design guideline by developing JooJ, an Eclipse plugin that provides real-time notification of violations of this guideline. In a broader context JooJ can be thought of as a poka-yoke approach to software quality assurance because it aims to prevent and detect violations of software design guideline, as or before they occur.

While we have established the feasibility of real-time cycle detection, determining its usability, that is, whether programmers will actually avoid dependency cycles, will require a higher quality implementation than the prototype we currently have. Producing such an implementation is currently underway. We would also like to expand JooJ to support other design principles that also require whole program analysis.

References

- Arthorne, J. & Laffra, C. (2004), *Official Eclipse 3.0 Faq (Eclipse Series)*, Addison-Wesley Professional.
- Binder, R. V. (1999), *Testing object-oriented systems: models, patterns, and tools*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

⁴<http://classycle.sourceforge.net/>

⁵<http://www.clarkware.com/software/JDepend.html>

- Bloch, J. (2001), *Effective Java programming language guide*, Sun Microsystems, Inc., Mountain View, CA, USA.
- Booch, G. (1995), *Object solutions: managing the object-oriented project*, Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.
- Booch, G. (1996), *Best of Booch: Designing Strategies for Object Technology*, SIGS Books.
- Briand, L. C., Daly, J. W. & Wust, J. (1998), 'A unified framework for cohesion measurement in object-oriented systems', *Empirical Softw. Engg.* **3**(1), 65–117.
- Briand, L. C., Labiche, Y. & Wang, Y. (2003), 'An investigation of graph-based class integration test order strategies', *IEEE Trans. Softw. Eng.* **29**(7), 594–607.
- Cormen, T. T., Leiserson, C. E. & Rivest, R. L. (1990), *Introduction to algorithms*, MIT Press, Cambridge, MA, USA.
- Eades, P., Lin, X. & Smyth, W. F. (1993), 'A fast and effective heuristic for the feedback arc set problem', *Inf. Process. Lett.* **47**(6), 319–323.
- Egyed, A. (2006), Instant consistency checking for the uml, in 'ICSE '06: Proceeding of the 28th International Conference on Software Engineering', ACM Press, New York, NY, USA, pp. 381–390.
- Fayad, M. E., Laitinen, M. & Ward, R. P. (2000), 'Thinking objectively: software engineering in the small', *Commun. ACM* **43**(3), 115–118.
- Foote, B. & Yoder, J. W. (2000), Big ball of mud, in N. Harrison, B. Foote & H. Rohnert, eds, 'Pattern Languages of Program Design', Vol. 4, Addison Wesley, pp. 654–692.
- Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995), *Design patterns: elements of reusable object-oriented software*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- Hashim, N. L., Schmidt, H. W. & Ramakrishnan, S. (2005), Test order for class-based integration testing of Java applications, in 'QSIC '05: Proceedings of the Fifth International Conference on Quality Software', IEEE Computer Society, Washington, DC, USA, pp. 11–18.
- Hautus, E. (2002), Improving Java software through package structure analysis, in 'The 6th IASTED International Conference Software Engineering and Applications'.
- Lakos, J. (1996), *Large-scale C++ software design*, Addison Wesley Longman Publishing Co. Inc., Redwood City, CA, USA.
- Martin, R. C. (1996), 'Granularity', *C++ Report* **8**(10), 57–62.
- Melton, H. & Tempero, E. (2006), An empirical study of cycles among classes in Java, in 'Tech. Report UoA-SE-2006-1', Department of Computer Science, University of Auckland.
- Melton, H. & Tempero, E. (2007), The CRSS metric for package design quality, in 'Thirtieth Australasian Computer Science Conference (ACSC2007)', Australian Computer Society, Inc.
- Parnas, D. L. (1978), Designing software for ease of extension and contraction, in 'ICSE '78: Proceedings of the 3rd international conference on Software engineering', IEEE Press, Piscataway, NJ, USA, pp. 264–277.
- Pressman, R. S. (2001), *Software engineering: a practitioner's approach (5th ed.)*, McGraw-Hill, Inc., New York, NY, USA.
- Riel, A. J. (1996), *Object-Oriented Design Heuristics*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- Robbins, J. E., Hilbert, D. M. & Redmiles, D. F. (1998), Software architecture critics in argo, in 'TUI '98: Proceedings of the 3rd international conference on Intelligent user interfaces', ACM Press, New York, NY, USA, pp. 141–144.
- Sangal, N., Jordan, E., Sinha, V. & Jackson, D. (2005), Using dependency models to manage complex software architecture, in 'OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN conference on Object oriented programming systems languages and applications', ACM Press, New York, NY, USA, pp. 167–176.
- Skiena, S. S. (1998), *The algorithm design manual*, Springer-Verlag New York, Inc., New York, NY, USA.
- Sun (1999), 'Code conventions for the Java programming language'.
URL: <http://java.sun.com/docs/codeconv/>

HAT-trie: A Cache-conscious Trie-based Data Structure for Strings

Nikolas Askitis

Ranjan Sinha

School of Computer Science and Information Technology,
RMIT University, Melbourne 3001, Australia.
Email: {naskitis,rsinha}@cs.rmit.edu.au

Abstract

Tries are the fastest tree-based data structures for managing strings in-memory, but are space-intensive. The burst-trie is almost as fast but reduces space by collapsing trie-chains into buckets. This is not however, a cache-conscious approach and can lead to poor performance on current processors. In this paper, we introduce the HAT-trie, a cache-conscious trie-based data structure that is formed by carefully combining existing components. We evaluate performance using several real-world datasets and against other high-performance data structures. We show strong improvements in both time and space; in most cases approaching that of the cache-conscious hash table. Our HAT-trie is shown to be the most efficient trie-based data structure for managing variable-length strings in-memory while maintaining sort order.

Keywords: String, tree, trie, hash table, cache, in-memory.

1 Introduction

Trie-based data structures offer rapid access to strings while maintaining reasonable worst-case performance. They are successful in a variety of applications, such as text compression (Bell, Cleary & Witten 1990) and dictionary management (Aoe, Morimoto & Sato 1992), but are space-intensive. Measures need to be taken to reduce their space consumption, if they are to remain feasible for maintaining large sets of strings.

The most successful procedure for reducing the space of a trie structure has been the burst-trie (Heinz, Zobel & Williams 2002). The burst-trie is an in-memory string data structure that can significantly reduce the number of trie nodes maintained by as much as 80%, at little to no cost in access speed. It achieves this by selectively collapsing chains of trie nodes into small buckets that share a common prefix. When a bucket has reached its capacity, it is *burst* into smaller buckets that are parented by a new trie node.

Buckets are internally unsorted, but they can be accessed in lexicographic order. The burst-trie can therefore provide fast sorted access to strings; buckets can be rapidly sorted on demand. It is currently the fastest and most compact in-memory tree structure that can handle large volumes of variable-length strings, in sort order. The effectiveness of the burst-trie has been demonstrated by its basis in burst-sort (Sinha, Ring & Zobel 2006, Sinha & Zobel 2004),

which is, to the best of our knowledge, the fastest in-memory data structure for the sorting of large sets of strings.

Although fast, the burst-trie is not cache-conscious. Like many in-memory data structures, it is efficient in a setting where all memory accesses are of equal cost. In practice however, a single random access to memory typically incurs many hundreds of clock cycles. Current processors therefore implement a hierarchy of small but fast caches between the CPU and main memory — that intercept and service memory requests whenever possible — to attempt to minimize the consultation of main memory (a cache-miss). It is of paramount importance to make the best possible use of these caches, to prevent severe performance bottlenecks that arise from excessive cache-misses. Programmers however, generally have no administrative control over these caches, yet this is usually not a problem in practice. Programs have been shown to make good use of cache through careful design that aims at improving *temporal* and *spatial* access locality. This is achieved through a variety of techniques that generally attempt to make memory access more regular or predictable.

Although space-intensive, tries can — to some extent — be cache-conscious. Trie nodes are small in size, improving the probability of frequently accessed trie-paths to reside within cache. The burst-trie however, represents buckets as linked lists which are known for their cache inefficiency. When traversing a linked list, the address of a child can not be known until the parent is processed. Known as the pointer-chasing problem, this hinders the effectiveness of hardware prefetchers (Yang, Lebeck, Tseng & Lee 2004), that attempt to reduce cache-misses by anticipating and loading data into cache ahead of the running program. This is the property underlying the efficiency of structures such as the cache-conscious array hash (Askitis & Zobel 2005), where the usual linked lists of a chaining hash table are replaced by dynamic arrays that are contiguously allocated in memory. Arrays exhibit stride access patterns that can be easily detected and exploited by hardware prefetchers.

Our motivation was to address the shortcomings of the burst-trie, by developing a cache-conscious variant that exploits the cache hierarchy used on modern processors. We introduce the *HAT-trie*, which combines the trie-index of the burst-trie with buckets that are represented as cache-conscious hash tables (Askitis & Zobel 2005). We present two variants of the HAT-trie that differ in the manner of how buckets are split: *hybrid* and *pure*. The former employs the B-trie splitting algorithm (Askitis & Zobel 2006) that reduces the number of buckets at little cost, by permitting multiple pointers to a bucket. The latter employs the *bursting* technique of the burst-trie, which is faster but usually requires more space.

Copyright ©2007, Australian Computer Society, Inc. This paper appeared at Thirtieth Australasian Computer Science Conference (ACSC2007), Ballarat, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 62. Gillian Dobbie, Ed. Reproduction for academic, not-for-profit purposes permitted provided this text is included.

We experimentally examined the performance of the HAT-tries, comparing them to a selection of data structures that are currently among the best in dynamic in-memory string management: the burst-trie, array hash, chained hash tables with move-to-front, and the *compact* binary search tree, which is a more cache-efficient variant that eliminates string pointers by storing strings within their respective nodes (Askitis & Zobel 2005). We used datasets acquired from real-world sources — containing variable-length strings with a range of characteristics — to compare space, time and cache efficiency of building and searching the data structures. To build and search, our HAT-tries were up to 80% faster than the burst-trie, with a (simultaneous) space reduction of up to 70%. The HAT-tries were more efficient than the optimized binary search tree, with the pure HAT-trie in particular, approaching, and in some cases surpassing, the performance of the cache-conscious array hash, which is currently the best for unsorted string management. These are strong results that further substantiate the importance of considering cache on pointer-intensive data structures, that are otherwise efficient.

2 Background

A trie node, named for its successful use in information retrieval (Fredkin 1960), is an array of pointers, one for each character in an alphabet. Each leaf node is the terminus of a chain of trie nodes representing a string (Knuth 1998). For string management, tries are fast with reasonable worst-case performance, and are valuable for applications such as data mining (Agrawal & Srikant 1994), dictionary and text processing (Aoe et al. 1992, Bentley & Sedgwick 1997), pattern matching (Flajolet & Puech 1986), and compression (Bell et al. 1990). Tries can be regarded as cache-conscious, as frequently accessed trie-paths may be cache-resident. Nonetheless, they are space-intensive, prohibiting use with large volumes of strings (Comer 1979, McCreight 1976).

Space can be conserved by reducing the number of trie nodes and by changing their structure. The compact trie (Bell et al. 1990) omits chains of tries that descend into a single leaf. Similarly, the Patricia trie (Sedgwick 1998) omits all chains without branches, not just those that lead to leaves. Tries can be represented as linked lists (Severance 1974), eliminating unused pointers. The ternary search tree (Bentley & Sedgwick 1997, Sedgwick 1998) can save space by using 3-way trie-nodes for less than, equal to, and greater than comparisons; reducing node size for sparse data. Trie compression (Al-Suwaiyel & Horowitz 1984), trie compaction (Maly 1976), and heuristics (Comer 1979) have also been applied. These techniques, however, save space at the expense of time and do not take the memory hierarchy into account. Acharya et al. (Acharya, Zhu & Shen 1999) addressed this issue by developing cache-efficient algorithms that choose between several representations of trie nodes. Although fast, space efficiency relative to previous methods is unclear for large sets of strings.

The burst-trie (Heinz et al. 2002) successfully reduced the number of trie nodes at little cost, by collapsing trie-chains into buckets that share a common prefix. Buckets — represented as linked lists with move-to-front on access (Knuth 1998) — are then selectively burst into smaller buckets that are parented by a new trie. Although computationally efficient, the burst-trie was not designed to exploit cache; linked lists are not cache-conscious structures (Yang et al. 2004). Consequentially, performance is jeop-

ardized when used on modern cache-oriented processors.

Current processors attempt to reduce memory latency by prefetching items that are likely to be requested in the near future. Hardware prefetchers (Collins, Sair, Calder & Tullsen 2002, Yang et al. 2004) generate prefetch requests from information accumulated at runtime. The simplest implementation being a hardware-based stride prediction table (Fu, Patel & Janssens 1992), which works well with array-based applications. Similarly, software prefetchers such as software caching (Aggarwal 2002) and jump-pointers (Roth & Sohi 1999), attempt to hide latency by fetching data before it is needed. Nonetheless, the effectiveness of prefetching remains poor when applied to pointer-intensive data structures. The binary search tree, where nodes and their associated strings are likely scattered in main memory, is an example.

Careful data layout (Chilimbi, Hill & Larus 1999) and cache-conscious memory allocation (Badawy, Aggarwal, Yeung & Tseng 2001, Hallberg, Palm, & Brorsson 2003) have shown the best results at improving the cache usage of pointer-intensive data structures. These techniques address the cause of cache misses, being poor access locality, rather than the manifestation of cache misses. The cache-sensitive search tree (Rao & Ross 1999) and the cache-conscious B⁺-tree (Rao & Ross 2000) are such examples, that exploit cache by changing the layout of nodes to eliminate (almost all) pointers to random memory. Nodes are contiguously allocated and are accessed through arithmetic offsets, improving spatial locality by making good use of hardware prefetchers (Berg 2002). These data structures are fast but remain expensive to update.

Based on the success of copying strings to array-based buckets in string sorting (Sinha et al. 2006), Askitis and Zobel (2005) replaced the linked lists of the chaining hash table (previously the best method for string hashing (Zobel, Williams & Heinz 2001)) with re-sizable buckets (arrays), forming the cache-conscious array hash. Arrays require more instructions for search, but compensate by eliminating the pointer-chasing problem through contiguous memory allocation. Performance gains of up to 96% over chained hashing were observed for both search and update costs, with space overheads at best, reduced to less than two bits per string, at no impact to speed. It is plausible that similar techniques can also lead to substantial gains for pointer-intensive data structures, such as the burst-trie.

Cache-oblivious data structures have received considerable attention in recent literature. These data structures aim to perform well on all levels of the memory hierarchy — including disk — without prior knowledge of the size and speed of each level (Frigo, Leiserson, Prokop & Ramachandran 1999, Kumar 2002). Brodal and Fagerberg for example, have recently proved the existence of a static (one that can not change once constructed) cache-oblivious string dictionary (Brodal & Fagerberg 2006). However, the study is from a theoretical perspective, and shows no empirical performance of speed and memory consumption, against well-known data structures; which is of value in practice. This is not uncommon, with studies demonstrating (in theory) that cache-oblivious structures can almost match the performance of optimized data structures (Bender, Brodal, Fagerberg, Ge, He, Hu, Iacono & Lopez-Ortiz 2003).

Similarly, the dynamic cache-oblivious B-tree (Bender, Demaine & Farach-Colton 2000) has been described in theory, but with no discussion of actual performance. The cache-oblivious dynamic

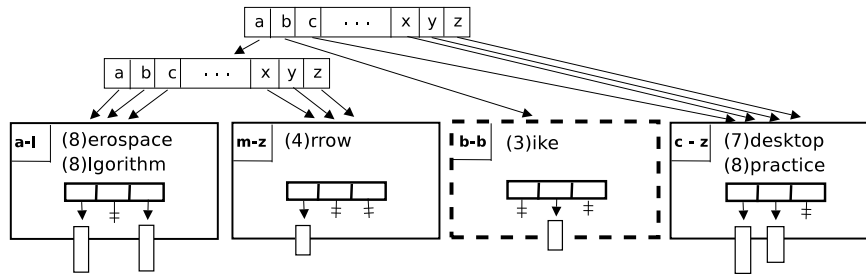


Figure 1: A hybrid HAT-trie, where buckets are split using B-trie splitting. A pure HAT-trie maintains only pure buckets that are burst, not split. Pure buckets are shown in dashed boxes. Strings are length-encoded (shown as brackets).

dictionary (Bender, Duan, Iacono & Wu 2002), which is a simplification of the cache-oblivious B-tree, is developed and evaluated against a B-tree, but only on a simulated memory hierarchy. Another example includes the cache-oblivious priority queue (Arge, Bender, Demaine, Holland-Minkley & Munro 2002). All of these data structures however, assume a uniform distribution in data and operations (Bender, Demaine & Farach-Colton 2002), which is typically not observed in practice.

There have been recent studies evaluating the practicality of these data structures (Ladner, Fortna & Nguyen 2002, Brodal, Fagerberg & Jacob 2002, Arge, Bender, Demaine, Leiserson & Mehlhorn 2004) with empirical results showing superior performance to conventional data structures, but not for those that have been tuned (both in memory consumption and time) to a specific memory hierarchy (Arge, Brodal & Fagerberg 2004). The data structures we present in this paper are not cache-oblivious, as they have been designed specifically to exploit cache between main memory and the CPU, and reside solely within (volatile) main memory. We omit comparisons of existing cache-oblivious data structures for our full paper.

3 The HAT-trie

The fastest data structures currently available for managing (storing and retrieving) variable-length strings in-memory, is the burst-trie (Heinz et al. 2002) and the chaining hash table with move-to-front on access (Zobel et al. 2001). These data structures are fast because they minimize the number of instructions required for search and update, but are not particularly cache-conscious. We know from literature that linked lists — used as buckets and as chains respectively — are the cause.

To address the bottlenecks of the burst-trie, we must significantly reduce the cost of trie traversal — being the number of trie nodes created — but more importantly, the cost of searching buckets, as these are currently the most expensive components to access. Such a reduction can only be achieved by changing the representation of buckets from linked lists, to large cache-conscious arrays. It is therefore attractive to consider structuring buckets as cache-conscious hash tables (Askitis & Zobel 2005). The advantage of using the array hash, as opposed to a simple array, is that buckets can scale much more efficiently, further reducing the number of trie nodes maintained.

This approach forms the basis of the HAT-trie, of which we studied two variants: *pure* and *hybrid*. These variants differ in the manner of how buckets are maintained and split. In the former, buckets contain strings that share only one prefix, which is removed.

These buckets are classified as *pure* and are referenced by a single parent pointer. In the latter, buckets also share a single prefix, however, it is not removed; the last character of the prefix is stored along with the string. These buckets are classified as *hybrid*, and have more than one parent pointer. The hybrid HAT-trie can also maintain *pure* buckets. An example is shown in Figure 1.

The HAT-tries are built and searched in a top-down manner. A trie is accessed by following the pointer corresponding to the lead character of the query. Pointer traversal will consume the lead character unless a hybrid bucket is accessed. Short strings can be consumed (deleted) and are handled by setting an end-of-string flag in the respective trie or pure bucket. When needed, an empty pure bucket can be created to serve as an end-of-string flag. Given the situation where strings have associated data fields (which does not arise in our data), alternative approaches to storing consumed strings is to use an auxiliary data structure (Askitis & Zobel 2006). Buckets can also be modified to accommodate data fields efficiently (Askitis & Zobel 2005).

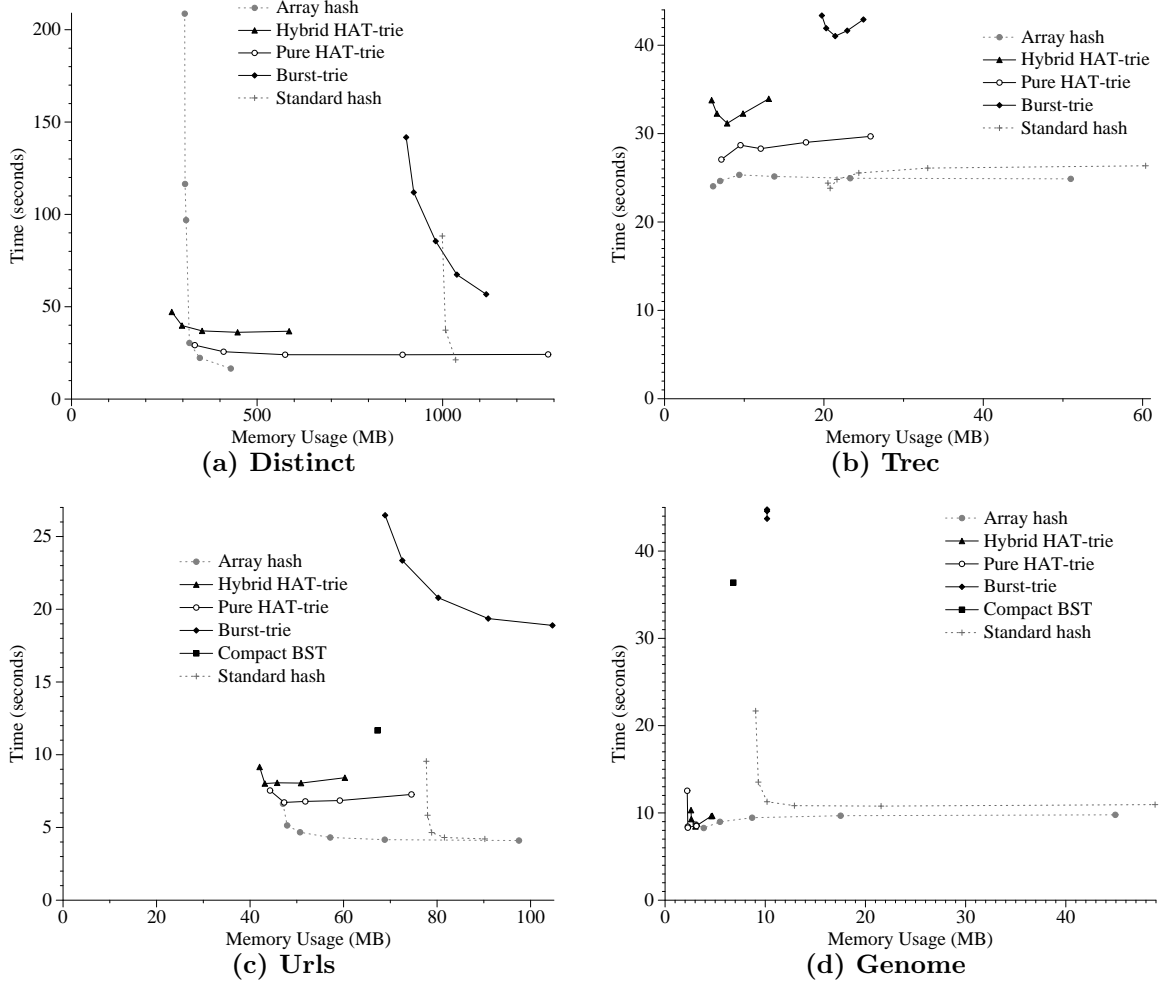
The HAT-trie begins as a single empty hybrid bucket, which is populated until full. Buckets do not maintain duplicates, hence an insertion occurs only on search failure. When a bucket is full, it is burst or split, depending on the type of HAT-trie. A pure HAT-trie, *bursts* a full bucket into at most A pure buckets that are parented by a new trie (A is the size of the alphabet). The strings are then distributed amongst the pure buckets, according to their lead character, which is removed. Strings can be consumed during the bursting procedure.

The hybrid HAT-trie however, *splits* buckets in two using the B-trie algorithm (Askitis & Zobel 2006), which we summarize. On split, a pure bucket is converted into a hybrid by creating a new parent trie with all pointers assigned to it. The old trie is pushed up as a grandparent and the splitting procedure continues as a hybrid. To split a hybrid, we find a suitable split-point that achieves good — preferably even — distribution, which may not always be possible. We access the strings in the bucket to accumulate the occurrence of every lead character. These occurrence counters are then traversed, in lexicographic order, to determine how many groups of strings to move, to acquire a distribution of at least 0.75 (number of strings moved divided by those that remain). This ratio was found to provide good space-efficiency for the b-trie. Once acquired, the split-point is chosen as the letter representing the current occurrence counter.

In the event where no strings remain, the last occurrence counter accessed is used as the split-point. This can result in the creation of an empty bucket. The two new buckets are then classified as pure or hybrid, based on split-point, and the strings are dis-

Dataset	Type	Distinct strings	String occs	Average length	Volume (MB) of distinct	Volume (MB) total
distinct	Text	28,772,169	28,772,169	9.59	304.56	304.56
trec	Text	612,219	177,999,203	5.06	5.68	1,079.46
urls	Non-text	1,287,597	9,997,487	30.93	45.82	318.56
genome	Non-text	262,084	31,623,000	9.0	3.8	316.23

Table 1: Characteristics of the datasets used in the experiments.

Figure 2: Construction costs of datasets *distinct*, *trec*, *urls* and *genome*. Graph plots represent bucket thresholds (sizes). Smaller buckets require more memory.

tributed accordingly. A string can be consumed during this process. Empty buckets are then deleted with their associated parent pointers set to null. The splitting procedure terminates only when both buckets do not exceed bucket capacity. Otherwise, the bucket that remains full is re-split.

Askitis and Zobel (2005) discuss in detail, the operations of the array hash. From an implementation perspective, we represent buckets as an array of $n + 1$ word-length pointers, which are empty or point to their respective slot entries; a dynamic bucket (array) containing length-encoded strings. We reserve the first pointer for house-keeping information: the bucket type or character range, the end-of-string flag and the number of strings.

To perform an insertion or search on a bucket, the required slot is found by first hashing what remains of the query string (after traversing the tries), by using a fast bitwise hash function (Ramakrishna & Zobel 1997). Once the required slot is accessed, its array is searched on a per-character basis, with a mismatch prompting a skip to the next string in

the array. Insertion only occurs when the string is not found or if the slot is empty. In such a case, the string is length-encoded and appended to the end of the array, by first growing (or initially creating) the array in an exact-fit manner, being the length of the string, which is cache-efficient.

The HAT-tries however, do impose a fixed space overhead per bucket, being the fixed number of slot pointers. Consequentially, some space is wasted when a bucket maintains only a few strings. It would be more efficient in this case, to represent an under-loaded bucket as a simple array which is changed to an array hash when full. Alternatively, buckets can maintain a variable number of slots, which is altered according to frequency of access. This will save space, but at the cost of re-hashing buckets.

4 Experimental Design

We compared the speed (*elapsed* time) of construction, search and the memory requirements of our HAT-tries against that of the array hash, the binary

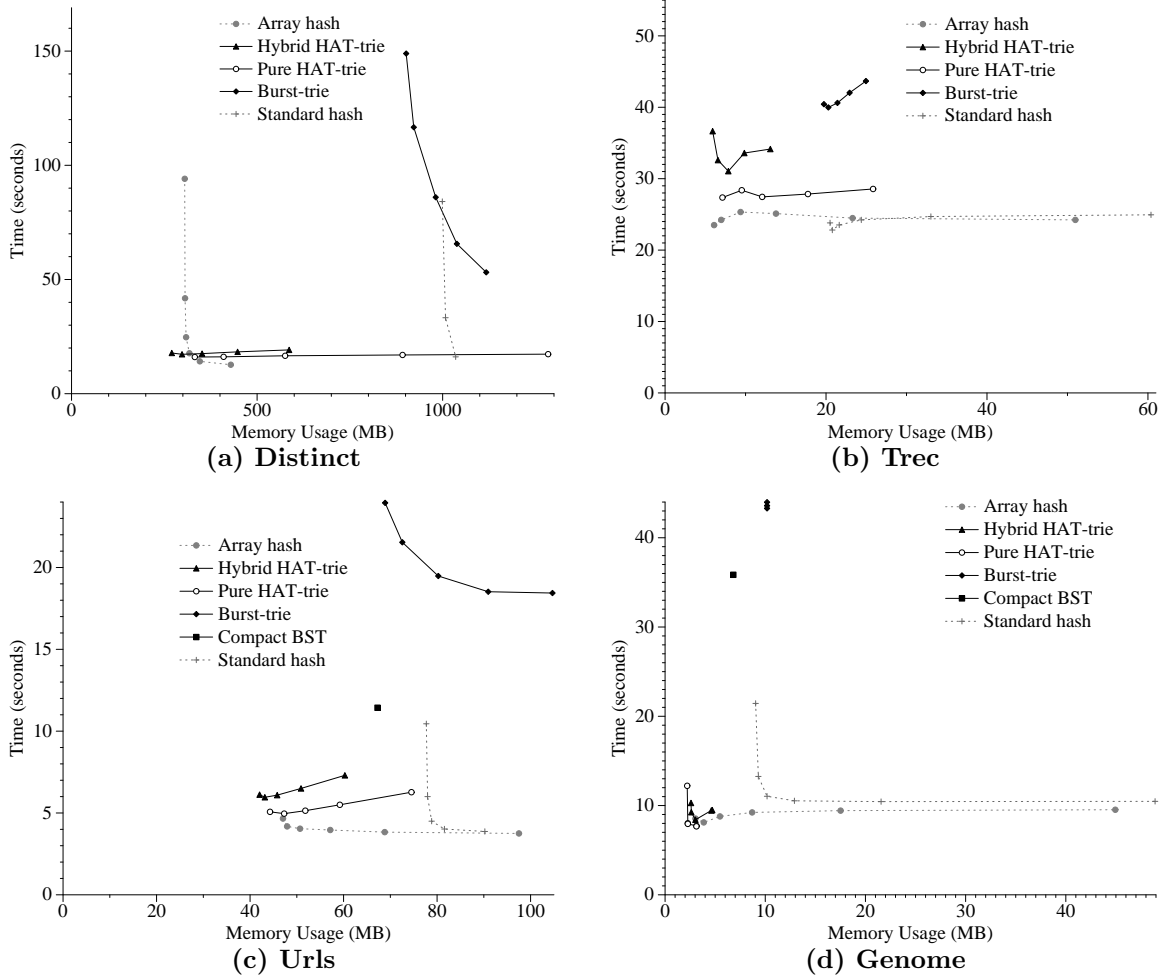


Figure 3: *Self-search costs of datasets distinct, trec, url and genome. Graph plots represent bucket thresholds (sizes). Smaller buckets require more memory.*

search tree, the burst-trie and the chaining hash table with move-to-front (which we call the *standard hash*); a group that includes the fastest and best-known data structures for in-memory string management. Our measure of memory takes into account the overhead imposed by the operating system per allocation, which is usually eight bytes; a figure which we found to be consistent after comparing our measure with that reported by the operating system under the */proc/stat/* table.

Although the binary search tree is well-known for its logarithmic average-case performance, it is not cache-conscious and in our initial experiments, we found it to be very slow. For a fairer comparison, we developed the *compact* BST, which stores strings directly within their respective nodes. This improves overall cache-efficiency (Askitis & Zobel 2005).

Our datasets consist of null-terminated variable-length strings that appear in occurrence order and are therefore unsorted; the characteristics of these datasets are shown in Table 1. The *distinct* dataset was acquired after parsing the GOV2 web crawl. Distributed as part of the TREC project, it does not contain duplicates. The highly skew *trec* dataset is the complete set of word occurrences, with duplicates, from the first of five TREC CDs (Harman 1995). The *url* dataset, also extracted from TREC web data, consists of complete URLs with duplicates. Finally, the *genome* dataset, extracted from GenBank, consists of fixed-length n-gram sequences with duplicates. Unlike the skew distributions observed in plain text however, these strings have a more uniform distribution.

The experiments were conducted on a 2.8 GHz Pentium IV machine, with 1 MB of L2 cache, 256-byte cache lines, a TLB capacity of 64 entries, 2 GB of RAM with a 4 KB page size, and a Linux operating system under light load using kernel 2.6.17. Our trie nodes used an alphabet that comprised of the first 128 characters of the ASCII table. We measured cache performance using PAPI (Dongarra, London, Moore, Mucci & Terpstra 2001) (available online), which obtains the actual number of TLB and L2 cache misses incurred during search. We restricted cache comparison to the array hash and the HAT-tries, as these were significantly better than the alternatives. We direct the reader to (Meyer, Sanders & Sibeyn 2003) for detailed information on the different types of caches used by modern processors.

The number of slots used by the array and standard hash were varied from 31,622 to 10,000,000 using the following sequence: 3.1622×10^4 , 10^5 , 3.16227×10^5 , 10^6 , 3.162277×10^6 , and 10^7 . For the burst-trie, we varied the bucket threshold (the number of strings needed to trigger a burst) by 25, 35, 50, 75, and 100. The HAT-tries have two parameters which are set at runtime: bucket threshold and the number of slots used. We began with a threshold of 1024 strings, doubling until 16,384. We set the number of slots used by every bucket to 1024 for the hybrid HAT-trie, and 512 for the pure HAT-trie. These settings were found to offer a good trade-off between space and speed. The pure HAT-trie requires fewer slots as it maintains only pure buckets that typically contain fewer strings.

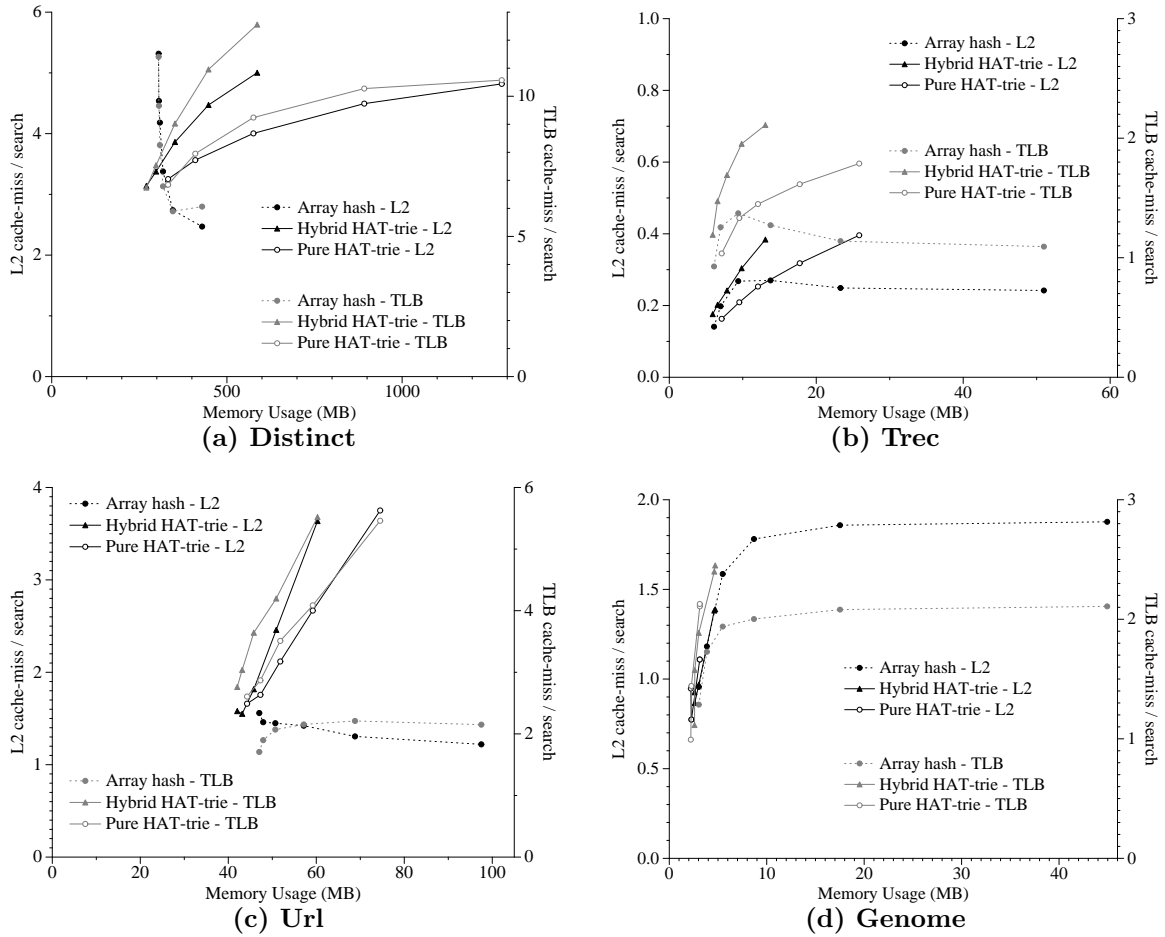


Figure 4: The number of L2 cache-misses during the self-search. Graph plots represent bucket thresholds (sizes). Smaller buckets require more memory.

5 Results

For all collections, we measured performance for construction and self-search. In a self-search, we search for all the words, in-occurrence order, that were used to construct the data structure. We now describe the results and observed trends for each collection.

Distinct data. The *distinct* dataset shows three important properties: there is no skew, each access requires a full tree traversal, and a large number of strings need to be managed. Such a collection is not particularly trie-friendly. Figures 2(a) and 3(a) show the effects on memory and speed as bucket thresholds vary. Small buckets are split often, resulting in a large number of tries and buckets. Larger buckets are split less frequently and are therefore, more space-efficient. We can see the effect on space in Table 2, which shows the number of nodes created relative to the bucket threshold.

We can save space by using larger buckets, but at the anticipated cost of access time. The HAT-tries however, showed little (if any) variance in the speed of construction and search, as the bucket threshold changed. For instance, with a threshold of 1024 strings, the hybrid HAT-trie required about 37 seconds to build and 19 seconds to search, consuming about 586 megabytes of memory. Buckets that were sixteen times larger increased the build time by less than ten seconds, while being faster to search (by about two seconds) and (simultaneously) reducing space to 270 megabytes; which is less than was required by the dataset. The pure HAT-trie displayed

even less variance in speed, with the largest buckets being just as fast to build and search, as smaller buckets. It remained consistently faster than the hybrid HAT-trie, as a result of bursting, which is cheaper and can lead to fewer strings per bucket, but at the cost of some space.

These results are in contrary to the burst-trie, which can only achieve its best time using smaller buckets, and best space with larger buckets. Either way, the burst-trie was of no contest, requiring over a gigabyte of memory to reach its optimal search time of 53 seconds. The hash tables remained the fastest data structures, but only when given enough space, which proved excessive for the standard hash, and was partially omitted in these graphs. In comparison however, the HAT-tries — the pure HAT-trie in particular — could almost match the speed and space-efficiency of the array hash, while maintaining sorted access to strings.

Despite our efforts at improving the cache-efficiency of the binary search tree, we omitted it in these graphs, as it required over 430 seconds to construct and search, using 764 megabytes of memory. Figure 4(a) shows the cache costs of the HAT-tries and the array hash during search. The array hash performed poorly under heavy load, incurring over five L2 cache misses and almost twelve TLB misses per search. As we add more slots to reduce the average load factor, buckets became smaller and cheaper to access, and have improved probability of cache residency. As a result, we see a sharp decline in the number of cache misses.

In contrast however, increasing memory usage (by using smaller buckets) is detrimental to the perfor-

	Slots	Threshold	Tries	Buckets	Memory (MB)
Hybrid HAT-trie	1,024	1,024	11,826	47,479	586.34
	1,024	2,048	6,384	23,374	447.48
	1,024	4,096	2,677	11,381	351.80
	1,024	8,192	1,283	5,762	297.73
	1,024	16,384	680	2,939	270.16
Pure HAT-trie	512	1,024	11,826	443,187	1,284.27
	512	2,048	6,384	264,529	891.97
	512	4,096	2,677	130,994	575.63
	512	8,192	1,283	64,001	409.72
	512	16,384	680	34,780	332.13
Burst-trie	-	25	564,723	6,153,817	1117.45
	-	35	406,799	5,021,979	1038.30
	-	50	292,000	4,058,240	981.30
	-	75	173,005	3,018,590	922.16
	-	100	128,868	2,498,127	901.74

Table 2: Node count as the bucket threshold varies with the *distinct* dataset.

mance of the HAT-tries. Although buckets remain cheaper to access when they are small in size, maintaining a large number of tries and buckets can ultimately reduce cache-efficiency, as fewer are likely to remain within cache. Furthermore, with an increase of trie nodes, cache will likely be flooded with tries on search, overwriting previously cached buckets. Consequentially, this will lead to an increase in *L2* cache misses, as observed.

Temporal access locality is unlikely to be as efficient for the HAT-tries, as it is for the hash tables. Every trie encountered during search branches to a new location in memory. Tries (apart from those close to the root of the tree) are therefore, less likely to be visited frequently, especially as their numbers increase. The number of conversions of virtual to physical addresses (*TLB* misses), is likely to increase as a result. Larger buckets are therefore of value to the HAT-tries, as they reduce the number of nodes created and space consumed, yielding higher cache efficiency.

Skewed data. Our next experiment evaluates cost where some strings are accessed much more frequently than others. We repeated the previous experiment using *trec*; the results are shown in Figures 2(b) and 3(b).

With just over half a million distinct strings, the data structures were much smaller than those constructed previously. The array hash performed well, even with a load factor of about 20 strings (31,622 slots). Although buckets were large, 99% of the searches need only access the first string of each bucket. Contrary to our previous results on the *distinct* dataset, it is clear that under skew access, the array hash can perform at its best, for both construction and search, with large buckets. The standard hash was also fast to access under heavy load as a result of move-to-front, but required more space.

Our HAT-tries remained competitive, being able to approach the speed of the array hash — the pure HAT-trie in particular — while almost matching its space consumption. The pure HAT-trie showed little variance in speed as we varied the bucket thresholds. The hybrid HAT-trie however, was slower to search with larger buckets, due to the use of b-trie splitting, which works more efficiently with a large number of strings per bucket (only 612,219 strings are distinct in this dataset). Both HAT-tries however, surpassed the performance of the burst-trie, which reached an optimal construction and search speed of about forty seconds (at least ten seconds slower) while requiring almost three times the space. The compact BST was

once again, too expensive and was omitted, requiring over 65 seconds to construct and search, while consuming about 15 megabytes of memory. It required less space than the burst-trie, as a result of eliminating string pointers.

Cache efficiency is shown with Figure 4(b). Larger buckets lead to a reduction in cache misses for the HAT-tries, which is consistent with what we observed in previous experiments. The array hash performed well with large buckets, as a result of heavy skew. That is, larger buckets (slot entries) are more frequently accessed and are likely to remain in cache longer than smaller, but more numerous buckets. The pure HAT-trie remains more cache efficient than the hybrid HAT-trie however, as a result of bursting buckets and using 512 (as opposed to 1024) slots per bucket, which can reduce *TLB* misses under skew. Although using large buckets is more cache-efficient, the cost of searching a hybrid HAT-trie with large buckets, is expensive. The cause is the manner of how buckets were split during construction. The b-trie algorithm is not as effective at splitting buckets, when deprived of strings (Askitis & Zobel 2006); buckets, although fewer in number, are likely to be split unevenly in this case.

This results in frequently accessed buckets that contain too many strings, and are therefore, more computationally expensive to search, albeit in cache. We confirmed this after comparing the instruction and CPU cycle costs during search, which were higher for larger buckets.

URL data. The *URL* dataset is highly skew and contains long strings, some in excess of a hundred characters. URLs typically share long prefixes; most URLs start with “http://www”. As a result, string comparisons can be more expensive.

The HAT-tries have the advantage of stripping away some of these prefixes during trie traversal, saving space and time. As a result, in Figures 2(c) and 3(c), the HAT-tries required the least amount of space (by using larger buckets) while simultaneously approaching the speed of the array hash. Interestingly however, the burst-trie remained expensive in both time and space. Even though prefixes are removed, the overhead imposed by linked list buckets, is too high.

The compact BST, although pointer-intensive, was almost twice as fast as the burst-trie, while requiring less space. This is due to the elimination of string pointers, which halves the number of random accesses made, the effects of which become more apparent under skew access. Its performance however, will start

to deteriorate as the number of searches increase, as observed with *trec*. As observed in previous experiments, the HAT-tries were able to perform at their best using large buckets that are more cache efficient, as shown in Figure 4(c).

Genome data. Our last experiment involves the *genome* dataset, containing fixed-length strings of strong skew. However, these strings are distributed much more uniformly than those of text. As shown in Figures 2(d) and 3(d), the HAT-tries almost matched the speed of the array hash for construction and search, while requiring less space. In contrast, for a high load factor, the cost of searching the array hash was higher, as string prefixes are not removed. Figure 4(d) shows that larger buckets allow the HAT-tries to approach and surpass the *L2* and *TLB* performance of the array hash. Combined with the use of a trie-structure that strips away common prefixes (saving both space and comparison costs), the HAT-tries were superior. The burst-trie and the compact BST were greatly inferior to the array hash and the HAT-tries. The standard hash could only approach their speed, once given excessive space.

6 Conclusion

The burst-trie is currently the fastest in-memory data structure for maintaining strings in sort order, which is for example, a key requirement needed by database management software. It is not however, cache-conscious which presents a serious performance bottleneck with modern processors that typically use a hierarchy of caches to reduce costly accesses to main memory. We have introduced the HAT-trie, a variant of the burst-trie that carefully combines existing data structures to yield a fast, compact, scalable, and cache-conscious data structure, that can maintain variable-length strings in-memory and in sort order. We proposed two versions of the HAT-trie, *hybrid* and *pure*, which differ in the manner of how they split nodes.

We compared the HAT-tries to the cache-conscious array hash (which is currently the best for unsorted string management), an optimized binary search tree, the burst-trie and the chaining hash table with move-to-front on access. For both construction and search, the HAT-tries were up to 80% faster than the burst-trie, while simultaneously reducing space consumption by as much as 70%. The chaining hash table could only compete in speed once given excessive space, while the binary search tree proved to be too inefficient in most cases.

Our HAT-tries — the pure HAT-trie in particular — could approach, and in some cases surpass, the speed of the array hash while requiring less space. In general, the hybrid HAT-trie required the least space of all data structures, but was not as fast as the pure HAT-trie. Our results highlight further potential improvements, primarily with the way buckets are structured. However, to our knowledge, the current HAT-tries are the first trie-based data structures that can approach the speed and space efficiency of hash tables, while maintaining sorted access to strings. These are strong results that further substantiate the effectiveness of using dynamic arrays in the structural design of pointer-intensive data structures that are otherwise computationally efficient.

References

- Acharya, A., Zhu, H. & Shen, K. (1999), Adaptive algorithms for cache-efficient trie search, in 'Proc. ALENEX Workshop on Algorithm Engineering and Experiments', Springer-Verlag, pp. 296–311.
- Aggarwal, A. (2002), Software caching vs. prefetching, in 'Proc. Int. Symp. on Memory management', ACM Press, New York, pp. 157–162.
- Agrawal, R. & Srikant, R. (1994), Fast algorithms for mining association rules, in 'Proc. VLDB Int. Conf. on Very Large Databases', Morgan Kaufmann, pp. 487–499.
- Al-Suwaiyel, M. & Horowitz, E. (1984), 'Algorithms for trie compaction', *ACM trans. on Database Systems* **9**(2), 243–263.
- Aoe, J., Morimoto, K. & Sato, T. (1992), 'An efficient implementation of trie structures', *Software—Practice and Experience* **22**(9), 695–721.
- Arge, L., Bender, M. A., Demaine, E. D., Holland-Minkley, B. & Munro, J. I. (2002), Cache-oblivious priority queue and graph algorithm applications, in 'Proc. ACM Symp. on Theory of Computing', ACM Press, New York, pp. 268–276.
- Arge, L., Bender, M. A., Demaine, E., Leiserson, C. & Mehlhorn, K. (2004), Abstracts collection, in 'Cache-Oblivious and Cache-Aware Algorithms', number 04301 in 'Dagstuhl Seminar Proceedings', IBFI, Germany.
- Arge, L., Brodal, G. & Fagerberg, R. (2004), In handbook on data structures and applications, in 'Cache-oblivious Data Structures', CRC Press.
- Askitis, N. & Zobel, J. (2005), Cache-conscious collision resolution for string hash tables, in 'Proc. SPIRE String Processing and Information Retrieval Symp.', Springer-Verlag, pp. 92–104.
- Askitis, N. & Zobel, J. (2006), B-tries for disk-based string management. Manuscript in submission.
- Badawy, A. A., Aggarwal, A., Yeung, D. & Tseng, C. (2001), Evaluating the impact of memory system performance on software prefetching and locality optimizations, in 'Proc. Int. Conference on Supercomputing', ACM Press, New York, pp. 486–500.
- Bell, T. C., Cleary, J. G. & Witten, I. H. (1990), *Text Compression*, Prentice-Hall.
- Bender, M. A., Demaine, E. D. & Farach-Colton, M. (2000), Cache-oblivious b-trees, in 'IEEE Symp. on the Foundations of Computer Science', IEEE Computer Society Press, pp. 399–409.
- Bender, M. A., Demaine, E. D. & Farach-Colton, M. (2002), Efficient tree layout in a multilevel memory hierarchy, in 'Proc. European Symp. on Algorithms', Springer-Verlag, pp. 165–173.
- Bender, M. A., Duan, Z., Iacono, J. & Wu, J. (2002), 'A locality-preserving cache-oblivious dynamic dictionary', *Proc. ACM-SIAM Symp. on Discrete Algorithms* **53**(2), 29–38.
- Bender, M., Brodal, G. S., Fagerberg, R., Ge, D., He, S., Hu, H., Iacono, J. & Lopez-Ortiz, A. (2003), The cost of cache-oblivious searching, in 'IEEE Symp. on the Foundations of Computer Science', IEEE Computer Society Press, pp. 271–282.
- Bentley, J. & Sedgwick, R. (1997), Fast algorithms for sorting and searching strings, in 'Proc. ACM-SIAM Symp. on Discrete Algorithms', Society for Industrial and Applied Mathematics, pp. 360–369.

- Berg, S. G. (2002), Cache prefetching, in 'Tech Report UW-CSE', Uni. of Washington.
- Brodal, G. S. & Fagerberg, R. (2006), Cache-oblivious string dictionaries, in 'Proc. ACM-SIAM Symp. on Discrete Algorithms', ACM Press, New York, pp. 581–590.
- Brodal, G. S., Fagerberg, R. & Jacob, R. (2002), Cache oblivious search trees via binary trees of small height, in 'Proc. ACM-SIAM Symp. on Discrete Algorithms', ACM Press, New York, pp. 39–48.
- Chilimbi, T. M., Hill, M. D. & Larus, J. R. (1999), Cache-conscious structure layout, in 'Proc. ACM SIGPLAN conf. on Programming Language Design and Implementation', ACM Press, New York, pp. 1–12.
- Collins, J., Sair, S., Calder, B. & Tullsen, D. M. (2002), Pointer cache assisted prefetching, in 'Proc. Annual ACM/IEEE MICRO Int. Symp. on Microarchitecture', IEEE Computer Society Press, pp. 62–73.
- Comer, D. (1979), 'Heuristics for trie index minimization', *ACM trans. on Database Systems* **4**(3), 383–395.
- Dongarra, J., London, K., Moore, S., Mucci, S. & Terpstra, D. (2001), Using papi for hardware performance monitoring on linux systems, in 'Proc. Conf. on Linux Clusters: The HPC Revolution', Urbana, Illinois, USA.
- Flajolet, P. & Puech, C. (1986), 'Partial match retrieval of multimedia data', *Jour. of the ACM* **33**(2), 371–407.
- Fredkin, E. (1960), 'Trie memory', *Communications of the ACM* **3**(9), 490–499.
- Frigo, M., Leiserson, C. E., Prokop, H. & Ramachandran, S. (1999), Cache-oblivious algorithms, in 'IEEE Symp. on the Foundations of Computer Science', IEEE Computer Society Press, p. 285.
- Fu, J. W. C., Patel, J. H. & Janssens, B. L. (1992), Stride directed prefetching in scalar processors, in 'Proc. Annual ACM/IEEE MICRO Int. Symp. on Microarchitecture', IEEE Computer Society Press, pp. 102–110.
- Hallberg, J., Palm, T., & Brorsson, M. (2003), Cache-conscious allocation of pointer-based data structures revisited with hw/sw prefetching, in '2nd Annual Workshop on Duplicating, Deconstructing, and Debunking'.
- Harman, D. (1995), Overview of the second text retrieval conference (TREC-2), in 'Proc. Second Text Retrieval Conference', Pergamon Press, Inc., pp. 271–289.
- Heinz, S., Zobel, J. & Williams, H. E. (2002), 'Burst tries: A fast, efficient data structure for string keys', *ACM trans. on Information Systems* **20**(2), 192–223.
- Knuth, D. E. (1998), *The Art of Computer Programming: Sorting and Searching*, Vol. 3, second edn, Addison-Wesley.
- Kumar, P. (2002), Cache oblivious algorithms., in 'Algorithms for Memory Hierarchies', Vol. 2625 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 193–212.
- Ladner, R. E., Fortna, R. & Nguyen, B. (2002), A comparison of cache aware and cache oblivious static search trees using program instrumentation, in 'Experimental algorithmics: from algorithm design to robust and efficient software', Springer-Verlag, pp. 78–92.
- Maly, K. (1976), 'Compressed tries', *Communications of the ACM* **19**(7), 409–415.
- McCreight, E. M. (1976), 'A space-economical suffix tree construction algorithm', *Jour. of the ACM* **23**(2), 262–271.
- Meyer, U., Sanders, P. & Sibeyn, J. F., eds (2003), *Algorithms for Memory Hierarchies*, Vol. 2625 of *Lecture Notes in Computer Science*, Springer-Verlag.
- Ramakrishna, M. V. & Zobel, J. (1997), Performance in practice of string hashing functions, in 'Proc. Int. Symp. on Database Systems for Advanced Applications', World Scientific Press, Melbourne, Australia, pp. 215–223.
- Rao, J. & Ross, K. A. (1999), Cache conscious indexing for decision-support in main memory, in 'Proc. VLDB Int. Conf. on Very Large Databases', Morgan Kaufmann, pp. 78–89.
- Rao, J. & Ross, K. A. (2000), Making b⁺-trees cache conscious in main memory, in 'Proc. ACM-SIGMOD Int. Conf. on the Management of Data', ACM Press, New York, pp. 475–486.
- Roth, A. & Sohi, G. S. (1999), Effective jump-pointer prefetching for linked data structures, in 'Proc. Int. Symp. on Computer Architecture', IEEE Computer Society Press, pp. 111–121.
- Sedgewick, R. (1998), *Algorithms in C*, Addison-Wesley.
- Severance, D. G. (1974), 'Identifier search mechanisms: A survey and generalized model', *Proc. ACM Computer Science Conf.* **6**(3), 175–194.
- Sinha, R., Ring, D. & Zobel, J. (2006), 'Cache-efficient string sorting using copying', *ACM Jour. of Exp. Algorithmics* **11**(1.2).
- Sinha, R. & Zobel, J. (2004), 'Cache-conscious sorting of large sets of strings with dynamic tries', *ACM Jour. of Exp. Algorithmics* **9**(1.5).
- Yang, C., Lebeck, A. R., Tseng, H. & Lee, C. (2004), 'Tolerating memory latency through push prefetching for pointer-intensive applications', *ACM Trans. Architecture Code Optimization* **1**(4), 445–475.
- Zobel, J., Williams, H. E. & Heinz, S. (2001), 'In-memory hash tables for accumulating text vocabularies', *Information Processing Letters* **80**(6), 271–277.

Web Services Discovery Based on Schema Matching

Yanan Hao

Yanchun Zhang

School of Computer Science and Mathematics
Victoria University
Melbourne, VIC, Australia
haoyan@csm.vu.edu.au
yzhang@csm.vu.edu.au

Abstract

A web service is programmatically available application logic exposed over Internet. With the rapid development of e-commerce over Internet, web services have attracted much attention in recent years. Nowadays, enterprises are able to outsource their internal business processes as services and make them accessible via the Web. Then they can dynamically combine individual services to provide new value-added services. A main problem that remains is how to discover desired web services. In this paper, we propose a novel web services discovery strategy given a textual description of services. In particular, we propose a new schema matching algorithm for supporting web-service operations matching. The matching algorithm catches not only structures, but also semantic information of schemas. We also propose a ranking strategy to satisfy a user's top-k requirements. Experimental evaluation shows that our approach can achieve high precision and recall ratio.

Keywords: Web service, XML Schema, Matching

1 Introduction

A web service is programmatically available application logic exposed over Internet. It has a set of operations and data types. The current set of web service specifications defines how to specify reusable operations through the Web-Service Description Language (WSDL) (Christensen, Curbera, Meredith & Weerawarana 2001), how these operations can be discovered and reused through the Universal Description, Discovery, and Integration (UDDI) API (Clement, Hatley, Riegen & Rogers 2004), and how the requests to and responses from web-service operations can be transmitted through the Simple Object Access Protocol API (SOAP) (Gudgin, Hadley, Mendelsohn, Moreau & Nielsen 2003).

With the rapid development of e-commerce over Internet, web services have attracted much attention in recent years. Nowadays, enterprises are able to outsource their internal business processes as services and make them accessible via the Web (see, e.g., (Wang, Zhang, Cao & Varadharajan 2003, Bhiri, Perrin & Godart 2005, Wang, Cao & Zhang 2005, Limthanmaphon & Zhang 2004, Limthanmaphon & Zhang 2003)). Then they can combine individual services into more complex, orchestrated services.

A main problem that remains is how to discover desired web services. To find a service in UDDI, a user

needs to input some keywords about the required service and then to browse the relevant UDDI category to locate relevant web services. Considering a large amount of service entries, this process is time consuming and frustrating. Furthermore, this method does not provide a mechanism assisting users in selection among similar web services. For example, consider the examples shown in Figure 1. A user searching for a *CreateOrder* service may also be interested in an *OrderGeneration* service. These two services are similar because they have the same function. But if the cost of *CreateOrder* is higher than that of *OrderGeneration*, the user would choose the latter one. This form of similarity potentially involves more web services. It is particularly useful and challenging in service composition.

This paper is devoted to address the problems above in web service search. The contribution of the work reported here is summarized as follows:

1. We propose algorithms for supporting web-service operations matching. The key part of our algorithms is a schema tree matching algorithm, which employs a new cost model to compute tree edit distances. Our new schema tree matching algorithm can not only catch structures, but also the semantic information of schemas.
2. Based on operations matching, we use the agglomeration algorithm to cluster similar web-service operations.
3. We also introduce a ranking strategy to satisfy a user's top-k requirements. Experimental evaluation shows that our approach can achieve acceptable result with high performance.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 gives an overview of our web service search approach. Section 4 describes a web-service operation matching algorithm, in which a new cost model and some XML schema transformation rules are defined. In section 5 we present how to cluster web-service operations. Section 6 describes our experimental evaluation. Section 7 gives some concluding remarks.

2 Related Works

Recently, several approaches have been proposed to find similar web services for a given web service. The earlier technique tModel presents an abstract interface to enhance service matching process. But the tModel needs to be defined while authors publishing in UDDI (Booth, Haas, McCab, Newcomer, Champion, Ferris & Orchard 2004). In (Sajjanhar, Hou & Zhang 2004), the authors propose a SVD-Based algorithm to locate matched services for a given service. This algorithm uses characteristics of singular

value decomposition to find relationships among services. But it only considers textual descriptions and can not reveal the semantic relationship between web services. Wang et al. (Wang & Stroulia 2003) proposed a method based on information retrieval and structure matching. Given a potentially partial specification of the desired service, all textual elements of the specification are extracted and are compared against the textual elements of the available services, to identify the most similar service description files and to order them according to their similarity. Next, given this set of likely candidates, a structure-matching method further refines the candidate set and assesses its quality. The drawback is that simple structural matching may be invalid when two web-service operations have many similar substructures on data types. Our approach is similar to this work, but we focus on the semantic similarity not the structural similarity. Woog (Dong, Halevy, Madhavan, Nemes & Zhang 2004) develops a clustering algorithm to group names of parameters of web-service operations into semantically meaningful concepts. Then these concepts are used to measure similarity of web-service operations. It relies too much on names of parameters and does not deal with composition problem however. (Shen & Su 2005) formally defines a behaviour model for web service by automata and logic formalisms. However, the behaviour signature and query statements need to be constructed manually, which can be very hard for common users.

3 An Overview of Web Services Search

The goal of our web-service search method is to find relevant web-service operations given a natural language description of desired web services and WSDL specifications of all available services published through UDDI. The WSDL files consist of textual description of web-service operations. Thus, firstly we use traditional IR technique TF (*term frequency*) and IDF (*inverse document frequency*) to find service operations that are most similar to the given description. We call these operations *candidate operations*. To do this, we extract words from web-service operation descriptions in WSDL. These words are pre-processed and assigned weight based on IDF. According to these weights, the similarity between the given description and a web-service operation description can be measured. A higher score indicates a closer similarity. For more details on measuring similarity among documents interested readers are referred to see (Salton, Wong & Yang 1975). After obtaining candidate operations, we employ a schema-match based method to measure similarity among them. Based on operations matching, the candidate operations are clustered into some *operation sets*. For each operation set the operation with the minimum cost in it is output as a search result. Since each candidate operation has a score, we can rank search results simply by the score of operations. Now we turn to the main focus of this paper, which is measuring similarity between web-service operations based on schema matching.

4 Web-service Operation Matching

4.1 Web-service Operation Modelling

Definition 1 A web service is a triple $ws = (TpSet, MsgSet, OpSet)$, where $TpSet$ is a set of data types; $MsgSet$ is a set of messages conforming to the data types defined in $TpSet$; $OpSet = \{op_i(input_i, output_i) | i = 1, 2, \dots, n\}$ is a set of operations, where $input_i$ and $output_i$ are param-

WS1: Web Service: CreateOrderService	
Operation: <i>OrderBuilder</i>	
Input: UserID	DataType: <i>int</i>
Output: ProductsList	DataType: <i>Order</i>
WS2: Web Service: OrderGeneration	
Operation: <i>GetOrder</i>	
Input: UserName	DataType: <i>String</i>
Output: MyProducts	DataType: <i>PurchaseOrder</i>

Figure 1: Sample Web-service Operations

ters(messages) for exchanging data between web-service operations.

Figure 1 gives two web-service operations used as examples in this paper. According to definition 1, a web service can be briefly described as a set of operations.

Definition 2 Each web-service operation is a multi-input-multi-output function of the form $f : s_1, s_2, \dots, s_n \rightarrow t_1, t_2, \dots, t_m$, where s_i and t_j are data types in according with XML schema specification. We call f a dependency and s_i/t_j a dependency attribute.

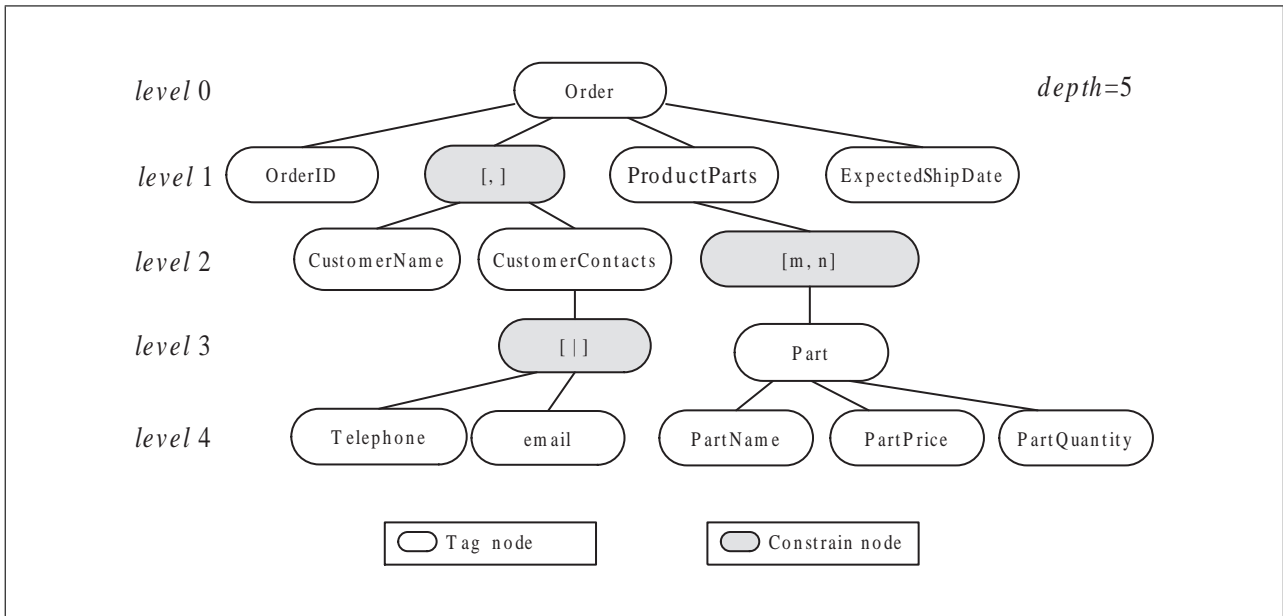
A dependency attribute can be a complex data type or a primitive data type. Complex data types, for example in *Order* and *PurchaseOrder* in Figure 1, define the structure, content, and semantics of parameters, whereas primitive data types, like *int* and *string*, are typically too coarse to reflect semantic information. We can convert primitive data types to complex data types by replacing them with their corresponding parameters. For example, in figure 1, *string* is converted into *UserName* type while *int* is converted into *UserID* type. Both *UserName* and *UserID* are considered as complex data types with semantics. Thus, each data type defined in a web-service operation carries semantic meaning.

An XML schema can be modelled as a tree of labelled nodes. We categorize a node n by its label:

1. **Tag node:** Each tag node n is associated with an element type T . T is also the tag name of node n .
2. **Constraint node:**
 - **Sequence node:** A sequence node indicates its children are an ordered set of element types. We use $[“,”]$ to denote a sequence node.
 - **Union node:** A union node represents a choice complex-type, that is, the instance of which can only be one of the children types in accordance with the XML Schema specification. We use $[“|”]$ to denote a union node.
 - **Multiplicity node:** Each node may optionally have a multiplicity modifier $[m, n]$ indicating that in the instance, its occurrence is between m and n . This corresponds to the *minOccurs* and *maxOccurs* constraints in XML Schema. We use $[m, n]$ to denote a multiplicity node.

As an example, the schema tree of data type *Order* is shown in Figure 2.

As we can see, data types defined in web-service operations carry semantic information. Intuitively,

Figure 2: XML schema tree of *Order* type

we consider two web-service operations similar if they have similar input/output data types. Thus the problem of web-service operation matching is converted to the problem of schema tree matching.

4.2 Tree Edit Distance

Many works have been done on the similarity computation on trees. Among them *tree edit distance* is one of the efficient approaches to describe difference between two trees. We introduce tree edit operations first. Generally, the tree edit distance operations include: (a) *node removal*, (b) *node insertion*, and (c) *node relabelling*. Such a set of operations can be represented by a mapping with minimum cost between the two trees. The concept of mapping is formally defined as follows (Reis, Golgher, d. Silva & Laender 2004):

Definition 3 Let T_x be a tree and let $T_x[i]$ be the i th node of tree T_x in a preorder traverse of the tree. A mapping between a tree T_1 and a tree T_2 is a set M of ordered pairs (i, j) , satisfying the following conditions for all $(i_1, j_1), (i_2, j_2) \in M$

1. $i_1 = i_2$ iff $j_1 = j_2$;
2. $T_1[i_1]$ is on the left of $T_1[i_2]$ iff $T_2[j_1]$ is on the left of $T_2[j_2]$;
3. $T_1[i_1]$ is an ancestor of $T_1[i_2]$ iff $T_2[j_1]$ is an ancestor of $T_2[j_2]$

Figure 3 gives an example of tree mapping. This mapping also shows the way of transforming the left tree to the right one. A dotted line from a node of T_1 to a node of T_2 indicates that the node of T_1 should be changed if the corresponding nodes are different, remaining unchanged otherwise. Nodes of T_1 not connected by dotted lines are deleted, and nodes of T_2 not connected are inserted.

Each of these operations is assigned a cost. The tree edit distance between two trees is defined as the minimal set of operations to transform one tree into the other.

Our schema matching algorithm is based on tree edit distance. However, the problem in our case is more complex than the traditional tree edit distance for the following reasons:

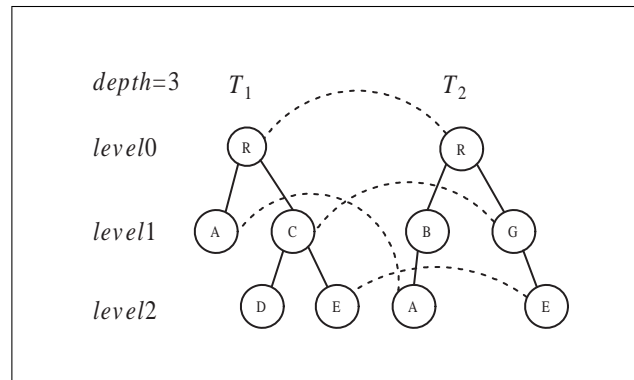


Figure 3: Example of tree mapping

1. The labels of an XML Schema tree can carry complex type information (e.g., union, multiplicity) which makes simple relabelling operations inapplicable. For instance, let T_1 and T_2 be the schema trees of *Order* and *Purchase-Order* respectively. Let us imagine there exists a mapping M between T_1 and T_2 , and there are two node-mapping pairs $(i_1, j_1), (i_2, j_2) \in M$, where $T_1[i_1] = [telephone | email]$, $T_2[j_1] = email$, $T_1[i_2] = price$, and $T_2[j_2] = quantity$. The edit operation of (i_1, j_1) should have less cost than that of (i_2, j_2) . But in the previous work, all tree edit operations are considered to have same unit distance.
2. The labels of nodes carry semantic information. So a relabelling from one node to another unrelated node will have more cost than to a semantic related node. For example, relabelling *part* to *item* is less costing than relabelling *price* to *email*.
3. We argue that tree edit operations on low-level nodes of a tree should have more influence than operations on high-level nodes. So, for example, if a *part* node on the third level of the first tree is mapped into a *part* node on the fifth level of the second tree, the edit operation cost should not be zero. But the traditional works on tree edit distance do not consider the difference and assign each edit operation unit cost.

In the next section, we present a new cost model to compute the cost of tree edit operation, as a consequence, the tree edit distance of two schema trees.

4.3 Cost Model

Measuring similarity between two XML schema trees equals to finding a mapping with minimum cost. So, the cost of each edit operation involved in the mapping needs to be computed first. In this section we introduce a new cost model based on tree edit distance presented in (Zhang & Shasha 1989) (Xie, Sha, Wang & Zhou 2006). The new cost model integrates weights of nodes and semantic connections between nodes. Let T_1, T_2 be two schema trees and let n , $node_1$ and $node_2$ be tree nodes. Formally, the cost model is defined as

$$cost(\rho) = \begin{cases} weight(n)/W(T_1, T_2), & \text{if } \rho = insert(n) \\ weight(n)/W(T_1, T_2), & \text{if } \rho = delete(n) \\ \alpha \times wd(node_1, node_2) & \text{if } \rho \text{ relabels} \\ +\beta \times sd(node_1, node_2) & \text{node}_1 \text{ to } node_2 \end{cases}$$

where ρ indicates a tree edit operation. $weight(n)$ shows the weight of node n . $wd(node_1, node_2)$ and $sd(node_1, node_2)$ give the weight and semantic difference of $node_1$ and $node_2$, respectively. α and β are weights of wd and sd , satisfying $\alpha + \beta = 1$. $W(T_1, T_2)$ is defined as $W(T_1, T_2) = weight(T_1) + weight(T_2)$, where $weight(T_i)$ is the sum of all node weights of tree T_i ($i = 1, 2$). $wd(node_1, node_2)$ is defined as

$$wd(node_1, node_2) = \frac{\|weight(node_1) - weight(node_2)\|}{W(T_1, T_2)}$$

where $node_1 \in T_1$ and $node_2 \in T_2$.

In the next two sections, we propose a set of schema-tree transformation rules and a semantic similarity measure to compute wd and sd , i.e. the weight and semantic difference of nodes.

4.4 XML Schema Tree Transformation

Definition 4 The tag name of a node is typically a sequence of concatenated words, with the first letter of every word capitalized (e.g., *ExpectedShipDate*). Such a set of words is referred to as a word bag. We use $\pi(n)$ to denote the word bag of node n .

Definition 5 Two word bags $\pi(n_1)$ and $\pi(n_2)$ are said to be equal, only if they have same words.

Two nodes are considered different if they have different word bags. The word bag reflects semantic meaning of a node. As we shall see later, using word bags we can measure the semantic similarity between two schema-tree nodes.

Definition 6 Let $level(n)$ denote the level of node n in schema tree T . The weight of node n is defined by a weight function:

$$weight(n) = 2^{depth(T) - level(n)} (\forall n \in T)$$

The weights of all nodes fall in the range of $[2, 2^{depth(T)}]$. Each weight reflects the importance of a node in schema tree T .

From section 4.2, it can be seen that traditional tree-edit-distance algorithm is not suitable for XML schema trees. It does not deal with constraint nodes. We propose three transformation rules to solve this problem. These rules are used to transform constraint nodes, specifically, sequence nodes, union nodes and multiplicity nodes to tag nodes. At the same time, the weights of nodes are reassigned.

1. *split*: This rule is applied to sequence nodes. A sequence node $l = [l_1, l_2, \dots, l_s]$ is split into an ordered list of nodes l_1, l_2, \dots, l_s , where l_i ($i = 1, 2, \dots, s$) is a child node of the sequence node l . After the split process, each sequence node is replaced by its child nodes. Each child node l_i inherits the weight of its parent node l as a new weight. Figure 4(a) gives an example of the split rule.
2. *merge*: This rule is applied to union nodes. After the merge process, each union node is replaced by all its option nodes, i.e. all its child nodes. All child nodes of the union node $l = [l_1|l_2|\dots|l_s]$ are merged into a new node l^* , while the union node l is deleted. The weight of node l^* is s times the weight of l . Each l_i 's ($i = 1, 2, \dots, s$) word bag is also merged into a new word bag. Formally, we have $weight(l^*) = weight(l) \times s$. Figure 4(b) gives an example of the merge rule.
3. *delete*: This rule is applied to multiplicity nodes. We delete a multiplicity node $l = [m, n]$ ($m, n \in N$) and scale up the weight of each of its child nodes l_i . After the deletion process, each multiplicity node is replaced by its child nodes. We have $weight(l_i) = weight(l) \times (m + n)/2$. Figure 4(c) gives an example of the delete rule.

Note that the definition of complex types can be nested according to XML schema specification. Thus, given a schema tree, we apply the three transformation rules to its nodes level by level, from bottom to top. This process is formally described as *bottom-up-transformation* algorithm (see Algorithm 1). The time complexity of Bottom-up-transformation is $O(n)$, where n is the number of nodes in the XML schema tree.

```

input : schema tree  $T$ 
output: transformed schema tree  $T^*$ 

1  $d = GetDepth(T)$ ;
2 for  $i \leftarrow d$  to 0 do
3   foreach node  $p \in level_i$  do
4     if  $p$  is a sequence node then
5       weight(each of  $p$ 's child
6         nodes)=weight( $p$ );
7       add  $p$ 's child nodes to  $p$ 's parent's
8         child list;
9       delete  $p$ ;
10    end
11    if  $p$  is a union node with  $s$  options
12       $\{l_i | i = 1, \dots, s\}$  then
13        merge  $p$ 's child nodes into a new
14        node  $q$ ;
15        add  $q$  to  $p$ 's parent's child list;
16        weight( $q$ ) = weight( $p$ )  $\times$   $s$ ;
17         $\pi(q) = \bigcup_{i=1}^s \pi(l_i)$ ;
18        delete  $p$ ;
19    end
20    if  $p$  is a multiplicity node  $[m, n]$  then
21      add  $p$ 's child node to  $p$ 's parent's
22      child list;
23      weight( $p$ 's child
24        node)=weight( $p$ )  $\times$   $(m + n)/2$ ;
25      delete  $p$ ;
26    end
27  end
28 end

```

Algorithm 1: Bottom-up-transformation

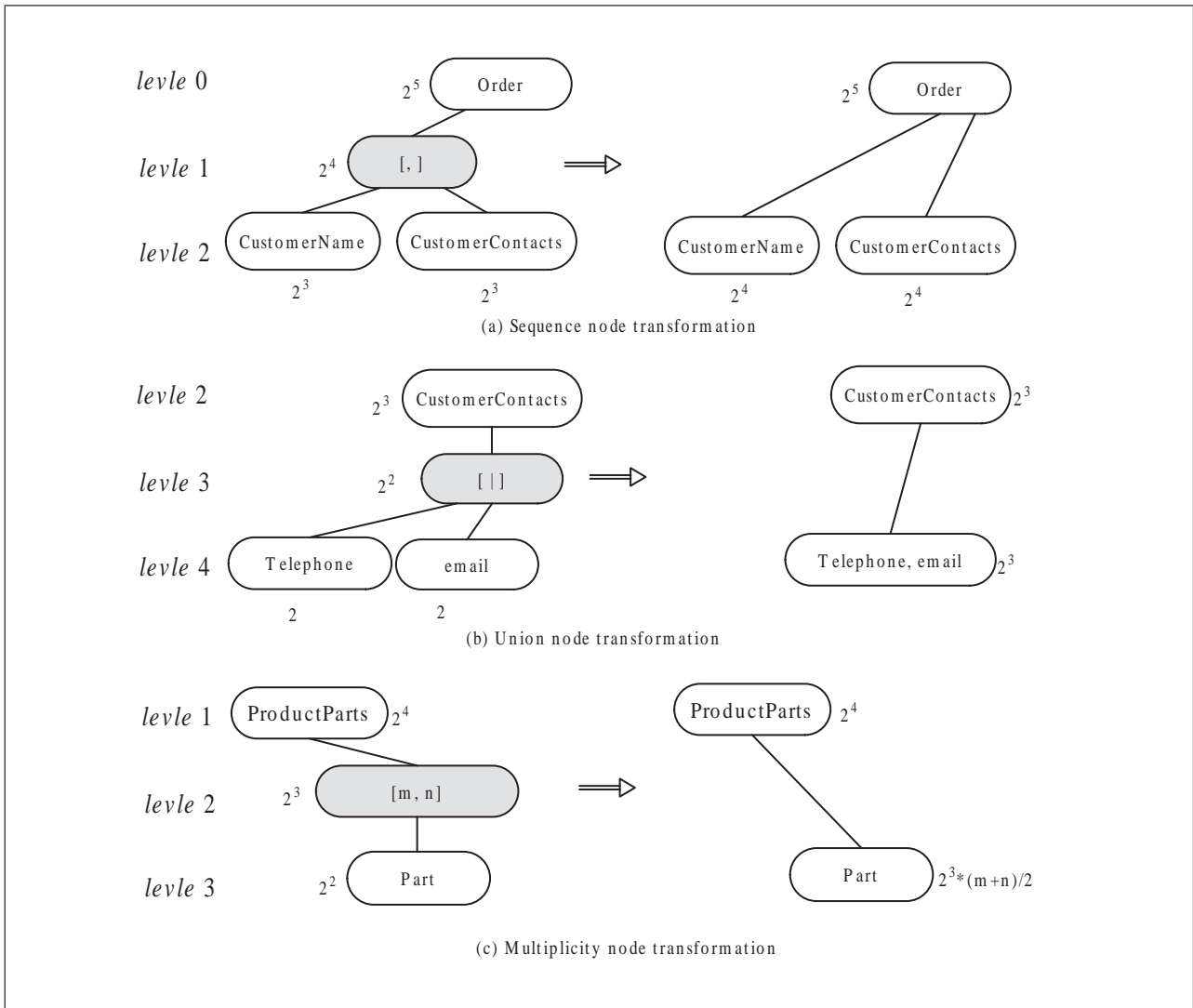


Figure 4: Examples of XML schema tree transformation

4.5 Semantic Measurement between Schema-tree Nodes

After the bottom-up transformation, schema tree T is converted into a new schema tree T^* . Each node n of T^* is a tag node, whose word bag may come from two or more word tags because of nodes mergence by the merge rule. Formally, node n can be regarded as a vector (W, B) , where W is the weight of node n and B is the word bag of node n . As we can see, after transformation the weight difference between two nodes can be computed by the new cost model. In this section, we present a strategy to determine the semantic similarity of two schema-tree nodes, i.e. the semantic distance between two word bags.

Our approach relies on a hypothesis that two co-occurrence words in a WSDL description tend to have same semantics. We exploit the co-occurrence of words in word bags to cluster them into meaningful concepts. To improve accuracy of semantic measurement, a pre-processing step is carried out first before words clustering. Pre-processing includes word stemming, removing stop words and expanding abbreviations and acronyms into the original forms.

Let $I = \{w_1, w_2, \dots, w_m\}$ be a set of words. These words come from word bags of all schema-tree nodes to which similarity measurement is applied. Let D be a set of candidate web-service operation descriptions available in WSDL files. We introduce association rules to reflect the notion of word co-occurrence.

An *association rule* is an implication of the form $w_i \rightarrow w_j$, where $w_i, w_j \in I$. The rule $w_i \rightarrow w_j$ holds in the descriptions set D with *support* s and *confidence* c , where s is the probability that w_i occurs in an web-service operation description; c is the probability that w_j occurs in an operation description, given w_i is known to occur in it. All association rules can be found by the A-Priori algorithm (Kaufman & Rousseeuw 1990). We are only interested in rules that have confidence above a certain threshold t .

We use the agglomeration algorithm (Kaufman & Rousseeuw 1990) to cluster words set $I = \{w_1, w_2, \dots, w_m\}$ into concept set $C = \{C_1, C_2, \dots\}$. There are three steps in the clustering process. It begins with each word forming its own cluster and gradually merges similar clusters.

1. Set up a confidence matrix $M_{m \times m}$. M_{ij} is a two-dimensional vector (s_{ij}, c_{ij}) , where s_{ij} and c_{ij} are the support and confidence of association rule $w_i \rightarrow w_j$, respectively.
2. Find M_{ij} with the largest c_{ij} in the confidence matrix M . If $c_{ij} > t$ and $s_{ij} > t$ then merge these two clusters and update M by replacing the two rows with a new row that describes the association between the merged cluster and the remaining clusters. The distance between two clusters is given by the distance between their closest members. There are now $m - 1$ clusters and $m - 1$ rows in M .

3. Repeat the merge step until no more clusters can be merged.

Finally, we get a set of concepts C . Each concept C_i consists a set of words $\{w_1, w_2, \dots\}$. To compute semantic similarity between schema-tree nodes, we replace each word in word bags with its corresponding concept, and then use the TF/IDF measure.

After schema-tree transformation and semantic similarity measure, the tree edit distance can be applied to match two XML schema trees by the new cost model.

4.6 Identifying Similar Web-service Operations

As it has been mentioned before, we use tree edit distance to match two schema trees. It is equivalent to finding the minimum cost mapping. Let M be a mapping between schema tree T_1 and T_2 , let S be a subset of pairs $(i, j) \in M$ with distinct word bags, let $D(I)$ be the set of nodes in $T_1(T_2)$ that are not mapped by M . The mapping cost is given by $C = Sp + Iq + Dr$, where p , q and r are the costs assigned to the relabel, insertion, and removal operations according to the cost model proposed in section 4.3. We call C the *match distance* between T_1 and T_2 , denoted as $C = ED(T_1, T_2)$. Match distance reflects semantic similarity of two schema trees.

Now let us see the algorithm for matching web-service operations. Given two web-service operations $op_1 : s_1, s_2, \dots, s_n \rightarrow t_1, t_2, \dots, t_m$ and $op_2 : x_1, x_2, \dots, x_l \rightarrow y_1, y_2, \dots, y_k$, we identify all possible matches between two lists of schema trees, and return the source-target correspondence that minimizes the overall match distance between the two lists. See Figure 5. We formally describe this process in algorithm 2.

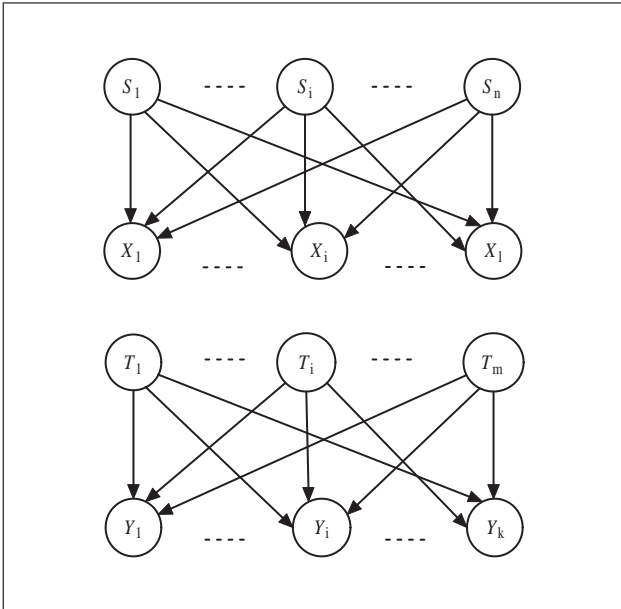


Figure 5: Matching Web-service Operations

5 Clustering Web-service Operations

Suppose $OP = \{op_1, op_2, \dots, op_q\}$ is a set of web-service operations and each pair of operations op_i and op_j ($i, j = 1, 2, \dots, q$) match with the distance of z_{ij} . We classify OP into a set of clusters $\{op_{c1}, op_{c2}, \dots\}$. The clustering algorithm is described as below. It begins with each operation forming its own cluster and gradually merges similar clusters.

```

input :  $op_1 : s_1, s_2, \dots, s_n \rightarrow t_1, t_2, \dots, t_m$ 
          $op_2 : x_1, x_2, \dots, x_l \rightarrow y_1, y_2, \dots, y_k$ 
output: The match distance  $Z$  between  $op_1$ 
         and  $op_2$ 

1 for  $i \leftarrow 1$  to  $n$  do
2    $S_i = \min\{ED(s_i, x_j) | j = 1, 2, \dots, l\}$ ;
3 end
4 for  $i \leftarrow 1$  to  $m$  do
5    $T_i = \min\{ED(t_i, y_j) | j = 1, 2, \dots, k\}$ ;
6 end

7  $Z = \sum_{i=1}^n S_i + \sum_{i=1}^m T_i$ 

```

Algorithm 2: Algorithm for matching web-service operations

1. Set up a match matrix $M_{q \times q}$. M_{ij} is the match distance of operation op_i and op_j .
2. Find the smallest M_{ij} in the match matrix M . If $M_{ij} < \text{threshold } \delta$ then merge these two clusters and update M by replacing the two rows with a new row that describes the association between the merged cluster and the remaining clusters. The distance between two clusters is given by the distance between their closest members. There are now $q - 1$ clusters and $q - 1$ rows in M .
3. Repeat the merge step until no more clusters can be merged.

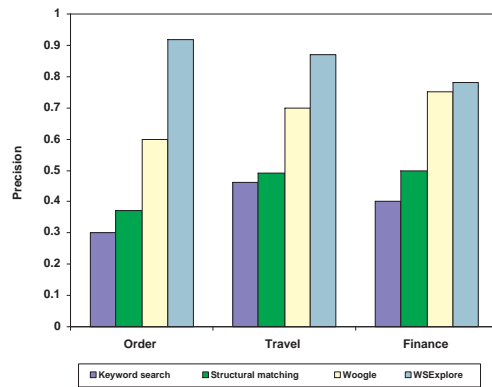
Finally, a set of clusters $\{OPC_1, OPC_2, \dots\}$ is obtained. For example, Figure 1 shows a sample cluster of two web-service operations: *GetOrder* and *OrderBuilder*. Given a cluster OPC_i and an operation $OPC_{ik} \in OPC_i$, OPC_{ik} is called the *pattern* of OPC_i if it has the minimum cost among OPC_i . We output all the patterns as search results.

6 Experiments and Evaluations

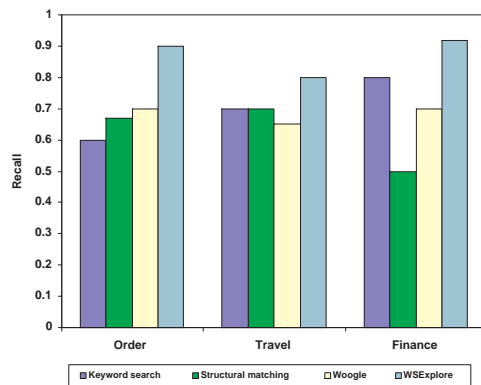
We have implemented a prototype system and conducted some experiments to evaluate the effectiveness and efficiency. We measured the efficiency of our web-service operation matching method by comparing it with keyword search, Woogle and structure matching. The experiments were conducted on a P4 Windows machine with a 2GHz Pentium IV and 512M main memory. The data set used in our tests is a group of web-service operations whose WSDL specifications are available, so we can obtain their textual descriptions and XML schemas of input/output data types. The data contains 223 web services including 930 web-service operations. We chose 7 web-service operations from three domains: *order*(3), *travel*(2) and *finance*(2). Each operation description was used as the basis for desired operations.

We use recall and precision ratio to evaluate the effectiveness of our approach. The precision(p) and recall(r) are defined as $p = \frac{A}{A+B}$, $r = \frac{A}{A+C}$ where A stands for the number of returned relevant operations, B stands for the number of returned irrelevant operations, C stands for the number of missing relevant operations, $A + C$ stands for the total number of relevant operations, and $A + B$ stands for the total number of returned operations. Specially, the top 100 search results are considered in our experiments for each web-service operation search.

We evaluated the efficiency of our approach by comparing the recall and precision of operation search with three other methods: keyword searching method, structure matching (Wang & Stroulia 2003) and Woogle (Dong et al. 2004). The results obtained



(a)



(b)

Figure 6: Precision and recall comparisons

are shown in Figure 6. As can be seen, the precisions of our approach are 92%, 87% and 78% respectively, almost always outperforming that of keyword, structure and Woogle. The precision is higher on *order* operations but lower in *finance* operations because *order* operations have more complex structures and richer semantics in input/output data types. This indicates that, by combining structural and semantic information, the precision of our approach improves significantly, compared to the results obtained with structural or semantic information only. It is also can be seen that by keyword method the precision is rather low whereas the recall is rather high. This demonstrates textual description of operations contain much useful information but also much noise at the same time.

7 Conclusions

In this paper we have presented a novel approach to retrieve desired web-service operations of a given textual description. The concept of tree edit distance is employed to match web-service operations. Meanwhile, some algorithms are proposed for measuring and grouping similar operations. Our approach can be used for web-service searching tasks with top-k requirements.

As part of on-going work, we are interested in improving performance of the web-service operation matching algorithm, as well as integrating more semantic information to our system in order to improve the search precision.

References

- Bhiri, S., Perrin, O. & Godart, C. (2005), Ensuring required failure atomicity of composite web services, in 'WWW', pp. 138–147.
- Booth, D., Haas, H., McCab, F., Newcomer, E., Champion, M., Ferris, C. & Orchard, D. (2004), Web services architecture. <http://www.w3.org/tr/ws-arch/>.
- Christensen, E., Curbera, F., Meredith, G. & Weerawarana, S. (2001), Web services description language (wsdl) 1.1. <http://www.w3.org/tr/wsdl>.
- Clement, L., Hatley, A., Riegen, C. V. & Rogers, T. (2004), Universal description discovery and integration. <http://uddi.org>.
- Dong, X., Halevy, A. Y., Madhavan, J., Nemes, E. & Zhang, J. (2004), Similarity search for web services, in 'VLDB', pp. 372–383.
- Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J. J. & Nielsen, H. F. (2003), Simple object access protocol (soap) version 1.2. <http://www.w3.org/tr/soap/>.
- Kaufman, L. & Rousseeuw, P. J. (1990), *Finding Groups in Data: An Introduction to Cluster Analysis*, John Wiley, New York. ID: 58.
- Limthanmaphon, B. & Zhang, Y. (2003), Web service composition with case-based reasoning., in 'ADC', pp. 201–208.
- Limthanmaphon, B. & Zhang, Y. (2004), Web service composition transaction management., in 'ADC', pp. 171–179.
- Reis, D. D. C., Golgher, P. B., d. Silva, A. S. & Laender, A. H. F. (2004), Automatic web news extraction using tree edit distance, in 'WWW', pp. 502–511.
- Sajjanhar, A., Hou, J. & Zhang, Y. (2004), Algorithm for web services matching, in 'APWeb', Vol. 3007, pp. 665–670.
- Salton, G., Wong, A. & Yang, C. S. (1975), 'A vector space model for automatic indexing', *Commun.ACM* **18**(11), 613–620.
- Shen, Z. & Su, J. (2005), Web service discovery based on behavior signatures, in 'SCC', Vol. 1, pp. 279–286 vol.1.
- Wang, H., Cao, J. & Zhang, Y. (2005), 'A flexible payment scheme and its role-based access control', *IEEE Trans. Knowl. Data Eng.* **17**(3), 425–436.
- Wang, H., Zhang, Y., Cao, J. & Varadharajan, V. (2003), 'Achieving secure and flexible m-services through tickets', *IEEE Transactions on Systems, Man, and Cybernetics, Part A* **33**(6), 697–708.
- Wang, Y. & Stroulia, E. (2003), Flexible interface matching for web-service discovery, in 'WISE'.
- Xie, T., Sha, C., Wang, X. & Zhou, A. (2006), Approximate top-k structural similarity search over xml documents, in 'APWeb', Vol. 3841, pp. 319–330.
- Zhang, K. & Shasha, D. (1989), 'Simple fast algorithms for the editing distance between trees and related problems', *SIAM J.Comput.* **18**(6), 1245–1262.

Conflict Management for Real-Time Collaborative Editing in Mobile Replicated Architectures

Sandy Citro, Jim McGovern, Caspar Ryan

School of Computer Science and Information Technology
RMIT University
Melbourne, Victoria

scitro@cs.rmit.edu.au, jim.mcgovern@rmit.edu.au, caspar@cs.rmit.edu.au

Abstract

Mobile technology is particularly suited to a fully distributed (replicated) architecture for collaborative work. Users can maintain their own document copies, and can continue to work in the absence of a central server. However, in a replicated architecture, conflicts can occur when two or more users concurrently modify the same object in a shared document. Such conflicts can be classified as *non-exclusive* or *exclusive*.

Non-exclusive conflicts, where conflicting operations can be realized at the same time, can be handled using conventional consistency management techniques such as operational transformation. On the other hand, *exclusive* conflicts can only be realised in different document versions. Although post-locking (Xue, Zhang, and Sun 2001) can be used to limit the number of versions that are created and thus reduce storage requirements in constrained mobile devices, it introduces two problems: a *partial-intention* problem and the need to synchronise locks before the conflict can be resolved.

This paper introduces an algorithm that integrates operational transformation and multi-versioning to resolve the different types of conflict. The algorithm uses *delayed post-locking* to solve the partial-intention problem by making use of *user intention locks*. It also uses *conflict tables* to better facilitate the resolution of conflict as soon as possible without requiring sites to receive all operations.

Keywords: Mobile Computing, Replicated Architecture, Collaborative Editing, Distributed Systems.

1 Introduction

Improved wireless technology and mobile devices have extended the ability to carry out collaborative activity to new situations and applications. Easily formed groups can carry out tasks such as search and rescue, military operations and to coordinate leisure activities. Key to these applications is the ability to communicate information and to construct shared artefacts. While collaborative editing of a range of document types has been widely researched in other technological domains, its extension to mobile and wireless environments is less

well understood. This paper explores flexible collaboration, in this environment by focussing on collaborative editing of an object-based document. In particular, the paper deals with conflict that can occur in a real-time editor based on a replicated architecture.

In a collaboration session, users edit the documents on their local devices and communicate with each other via message passing. Each user interacts with the shared document, as it appears at his/her local device, with changes propagated to other users as soon as possible to reduce the possibility of update conflicts (Citro, McGovern, and Ryan 2005). In a mobile network environment with nondeterministic communication latency, a replicated architecture is usually adopted for the storage of the shared document in order to provide high responsiveness (Dewan 1999, Xue, Zhang, and Sun 2001). The ability to work without a central server, and to operate during disconnection, makes the replicated architecture attractive for real time mobile group editors (Citro, McGovern, and Ryan 2005).

A conflict occurs when two or more users have different intentions for editing the same part of the replicated document. Note that in practice, the definition of a conflict is application dependent, with possible factors being domain specific semantics, implementation details, document granularity and desired level of concurrency. Nevertheless, in the general case, in an object based document, a conflict occurs when two or more users are concurrently modifying the same object (object-based conflict).

An object in a document, however, may comprise other objects. Therefore, conflicts may be defined in different levels of the object hierarchy. For example, conflict might only occur when the object being modified is the lowest in the object hierarchy. To further promote concurrency, a conflict can be defined to occur only when operations are concurrently modifying the same attribute of the same object to different values (attribute-based conflict) (Sun and Chen 2002). This paper uses a generic conflict definition: a conflict occurs when two or more users concurrently modify the same target with different intentions. The target is application-specific and could be an object, an object attribute, a word in a document, a letter in a document, a paragraph, or even a whole document.

Conflict can be categorised into two types: *exclusive* and *non-exclusive* conflicts. An *exclusive* conflict occurs when the conflicting operations cannot be realised at the same time, and if serially executed, the effect of the later operation will override the earlier operation. In contrast, a

non-exclusive conflict occurs when the conflicting operations can be realised at the same time and both operations can be applied to the target without one being overridden by the other. Suppose *Alice* and *Bob* are currently collaborating on an object-based document, with words being the targets, and they are concurrently modifying a word "and". *Alice* is adding a letter 'h' to make the word "hand" while *Bob* is changing the letter 'd' to 't' to make the word "ant". The operations are conflicting as they have different intentions, but they can be realised at the same time resulting with the word "hant". The word can then be marked as being in conflict and the conflict can then be resolved. This is an example of a *non-exclusive* conflict. However, if *Alice* is changing the font size of the word to size 12 and *Bob* is changing the font size to size 14, an *exclusive* conflict occurs. Both operations cannot be realised together, and one operation will override the other depending on the execution order. In other words, an exclusive conflict happens if the operations causing the conflict are *non-transformable* against each other.

Transactional processing has been a major topic in dealing with concurrent updates to a document/database (Coulouris, Dollimore and Kindberg 2001). However, transactional processing uses centralised server and it is used mainly in database application where users *read* the document from the server before they make changes and *write* back to the server. Transactional concurrency control focuses on grouping the operations together into an *atomic* transaction and serializing the transactions. However, in real time collaborations, each operation is processed as it arrives at the local replica so as to promote concurrency.

On the other hand, in replicated architecture real-time group editor research, the approaches adopted by the existing consistency management algorithms, including conflict resolution, can be categorised into: (1) locking approaches (Munson and Dewan 1996, McGuffin and Olson 1992, Newman-Wolfe, Webb and Montes 1992), (2) operational transformation approaches (Ellis and Gibbs 1989, Suleiman, Cart and Ferrié 1998, Vidot, Cart, Ferrié and Suleiman 2000, Sun, Jia, Zhang, Yang and Chen 1998), and (3) multi-versioning approaches (Sun and Chen 2002, Chen and Sun 1999). The locking approach is a conflict prevention approach rather than a conflict resolution approach, and it does not promote concurrency as only one person can modify an object at one time. The operational transformation approach is optimistic and is able to handle non-exclusive conflicts by transforming concurrent operations. However, the operational transformation approach in itself cannot handle exclusive conflicts. The existing operational transformation based algorithms serialise concurrent operations, including the conflicting ones, whereby the operation executed later will override the one executed earlier causing the intention of one or more users to be nullified. The multi-versioning approach handles exclusive conflicts by creating different object versions with each version realising each conflicting intention. This approach is attractive as it preserves the intentions of all users and can be used in various conflict resolution

strategies, such as voting or priority based authorization (Xue, Zhang and Sun 2000).

Xue, Zhang, and Sun (2001) combine multi-versioning, operational transformation, and post-locking (Xue, Zhang and Sun 2000) to handle both types of conflict, and to restrict the number of object versions. Post-locking involves the currently edited object being automatically locked whenever a conflict occurs, and the user might not have fully realised his/her intention on the target object. While post-locking simplifies the conflict resolution process by locking versions to protect them from further modification, it suffers from a *partial intention* problem insofar as conflicts could be better resolved if the conflicting intentions have been completely realised and thus every user can see the full intention of all other users. Furthermore, the existing post-locking approach requires the lock to be synchronized before conflicts can be resolved meaning that in mobile network environments, which are characterised by frequent disconnections, sites might have to wait indefinitely for the lock to be synchronized.

This paper introduces an algorithm that addresses the shortcomings of existing work using the following techniques. Firstly, *delayed post-locking* allows users to completely generate all operations necessary to realise their intention, before conflicts are established. Secondly, the algorithm allows users to resolve conflict as soon as possible without having to wait for the lock-synchronization period. Thirdly, *conflict tables* are introduced in order to: allow conflicts to be better organised; support a simpler and more flexible conflict resolution process; and keep users better informed of the status of the conflict. Finally, the algorithm is able to support group editing of any object based document, even those with internal object based dependencies. In contrast, most existing algorithms are applied either to plain text editors (with very simple primitives, such as insert and delete) or simple graphic editors based on independent objects.

The rest of the paper is structured as follows: Section 2 provides a brief review of the existing work on consistency and conflict management. Section 3 provides some basic background on object based document editing, including operation primitives and the transformation function. Section 4 describes the proposed conflict management algorithm. Finally, Section 5 concludes the paper and outlines possibilities for future work.

2 Related Work

2.1 Locking

Locking is a pessimistic approach which prevents conflicts in distributed systems by prohibiting concurrent updates on shared data objects. Locking has also been applied in various group editors for consistency management (Munson and Dewan 1996, McGuffin and Olson 1992, Newman-Wolfe, Webb and Montes 1992). However, locking is undesirable for the following reasons. Firstly, it imposes overheads in the lock requesting, granting and releasing procedures, especially in replicated architectures where there is no machine

dedicated to lock management. Secondly, it diminishes concurrency since users cannot modify the locked part of the document. Finally, the locking technique itself has not prevented divergence from occurring in a document where the objects are not independent (Sun 2002).

More optimistic locking strategies such as optimistic shared-locking (Sun and Sosič 1999) and tentative optional locking (Sun 2002) have also been proposed. Their non-blocking property allows users to continue updating the document while waiting for the lock. However, eventually they still need to wait for the lock to be resolved (if there are concurrent locks on the same region) to know whether their updates are to be kept or to be undone. Furthermore, they require additional operational transformation based rules for the locking operations to make sure the concurrent locking operations are applied consistently at all sites.

A number of other optimistic conflict resolution strategies, which are not based on locks, have also been proposed. These include operational transformation (Ellis and Gibbs 1989), multi-versioning (Sun and Chen 2002) and post-locking (Xue, Zhang, and Sun 2001). In contrast to the traditional locking, they do not prevent conflicts but instead resolve conflicts once they occur, using different approaches for different types of conflict. These strategies are described in the following three subsections.

2.2 Operational Transformation

Operational transformation was first introduced by Ellis and Gibbs (1989) in the dOPT algorithm to allow concurrent updates on document replicas. Operational transformation has been proven by Sun, Jia, Zhang, Yang and Chen (1998) to possess three consistency properties: convergence, causal preservation and intention preservation (albeit at the expense of semantic correctness). It preserves causality by implementing vector clocks, and preserves user intention by transforming concurrent operations consistently at all sites thereby enforcing document convergence. Most operational transformation based algorithms (Suleiman, Cart and Ferrié 1998, Vidot, Cart, Ferrié and Suleiman 2000, Sun, Jia, Zhang, Yang and Chen 1998, Citro, McGovern, and Ryan 2005) serialise concurrent operations: concurrent operations are executed sequentially with the later operations being transformed to include the effect of the earlier operation according to a certain total order scheme. They use different strategies to serialise the concurrent operations: undo/do/redo (Sun, Jia, Zhang, Yang and Chen 1998), history separation (Suleiman, Cart and Ferrié 1998), global sequencer (Vidot, Cart, Ferrié and Suleiman 2000) and distributed total ordering schemes (Citro, McGovern, and Ryan 2005). Unfortunately, they are only capable of handling non-exclusive conflicts. By serialising the concurrent operations, non-exclusive conflicts are preserved and the operations are applied to the document even though doing so may create errors in the document state. The users can then resolve the error by deleting one character or undoing an operation. Serialising exclusive conflicting operations means the operation executed later overrides

the effect of the operation executed earlier thereby clearly violating the intention of at least one user.

2.3 Multi-versioning

As an alternative to serialising conflicting operations using operational transformation, a multi-versioning approach is used to preserve the effect of both conflicting operations by creating multiple versions of the document and executing the operations in parallel, applying each operation to the different versions of the document (Sun and Chen 2002, Chen and Sun 1999). This approach is able to handle exclusive conflicts by preserving both operations; hence users are allowed to see the different conflicting intentions. Unfortunately, this approach has drawbacks, especially for mobile environments.

Firstly, by creating multiple versions of the document, it increases the required storage space, which is relatively limited in mobile devices. Secondly, a sophisticated object version identification scheme is required to ensure the operations are applied to the correct versions. This increases processing requirements which is undesirable in a mobile environment. Finally, the approach presented by Sun and Chen (2002) is applicable only to independent-object documents i.e. documents where one object's attribute or position does not affect other objects in the document.

2.4 Post Locking

In an effort to restrict the number of versions created in a multi-versioning approach, post-locking has been used in systems such as POLO (Xue, Orgun and Zhang 2003) and LOVOT (Xue, Zhang, and Sun 2001). POLO is applicable to independent-object documents, whereas LOVOT is applicable to dependent-object documents. Whenever a site receives a conflicting operation, once the object versions are created they are locked locally by the system until the conflict is resolved, with the intention that the user can modify other objects in the meantime. Note that the object versions are not necessarily post-locked at the same time at all sites since the post-lock is not propagated to other sites, but rather invoked on a case by case basis by the individual sites as they receive the conflicting operations. While this simplifies the conflict resolution process by locking the versions, it suffers from a partial intention problem. That is, the object being modified is automatically locked whenever a conflict occurs, and the user might not have fully realised his/her intention on the target object.

Furthermore, post locking relies on a lock synchronization process. The conflict resolution procedure can only be invoked if the locks are synchronized, meaning that all sites have received all conflicting operations and thus have access to the document versions created by those operations. Due to concurrency and network latency, the locks might not be synchronised at the same time at all sites and thus the conflict resolution procedure might never occur if one or more sites miss even a single conflicting operation.

2.5 Conflict Resolution Strategies

Unlike operation transformation which arbitrarily serialises conflicting operations, multi-versioning

approaches, such as the one presented in this paper, resolve user intention conflicts by creating document or object versions. Consequently, users must eventually decide which version is to be selected as the correct group-intended version.

A simple way of resolving conflict is priority based authorization, which gives specific members such as a group leader or administration group the right to resolve conflicts. Although this approach reduces network usage due to messages round-trips, it introduces additional points of failure and may not be the most effective working strategy from a usability point of view (Xue, Zhang and Sun 2000).

Alternatively, conflicts can be resolved by negotiation (Chu-Carroll and Carberry 1995) or reaching consensus amongst the group. For example, voting (Xue, Zhang and Sun 2000) aims to reach a group intended version where the decision is supported by at least the majority of users. Implementing such strategies in a distributed system is not easy, especially in a peer-to-peer mobile network environment (Fischer, Lynch and Paterson 1985). In particular, such strategies require complex semantics, consume considerable bandwidth, and require good connectivity among devices. Furthermore, negotiation may continue indefinitely if a consensus or suitable outcome is never reached.

3 Example - An Object Based Group Editor

This section describes a simple group editor, which is implemented using an object based text document, as an example application for the purpose of explaining the algorithm presented in this paper. The application domain is simple enough to provide an understanding of collaborating on object based documents and yet broad enough in scope to show the various kinds of conflicts and how they are dealt with. This section describes the basics of the object based document, including the operations supported and the impact of the conflict handling.

The text document consists of *character* objects that are identified by an object identifier (*objId*) and have a number attributes. Note that in a real application, each object in the document could have many attributes, however for the purpose of this paper, each object has the following attributes: position in the document (*pos*), size (*fontSize*) and colour (*fontColor*).

The *pos* attribute dependent on the position of other objects and thus the insertion or deletion of one object might affect the position of another, meaning that this is a *dependent object document*. Changing the other attributes, however, does not affect other objects and thus the document also shows independent object characteristics, thereby allowing the conflict management algorithm presented in this paper to be tested for both dependent- and independent-object cases. A series of adjacent character objects forms a *word* object. Non-exclusive conflicts happen when users are inserting different characters into the same word, and exclusive conflicts happen when users are changing the attributes size and color to different values. Note that a real document could have many other object structures such as paragraphs, phrases or sentences; however these are

not necessary within the scope of this paper to illustrate the functioning of the algorithm on a generalised object based document

The operation primitives used in the document are: *insert(objId, pos, char, attrSet)*, *delete(objId)*, and *changeAttr(objId, attr, value)*. An *insert* operation inserts a new character with a unique object id at a certain position in the document. The newly inserted character object has a set of attributes to determine its size and its color. A *delete* operation simply deletes a character identified by its object id. A *changeAttr* operation changes an attribute of a character to a certain value (eg. *changeAttr(X, fontSize, 16)*).

To preserve causality, each operation is timestamped with a state vector (SV_{op}) to signify the state of the originator site when the operation is generated (Ellis and Gibbs 1989).

Definition 1. Causality Relation

Let op_1 and op_2 be operations generated by site i and site j consequently. Operation op_1 causally precedes op_2 , $op_1 \rightarrow op_2$, iff $SV_{op_2}[i] > SV_{op_1}[i]$.

Definition 2. ConcurrencyRelation

Operations op_1 and op_2 are concurrent, $op_1 \parallel op_2$, iff $op_1 \not\rightarrow op_2$ and $op_2 \not\rightarrow op_1$

Non-exclusive conflicts involve concurrent *insert* and/or *delete* operations as they are always *transformable* as long as they are concurrent. Exclusive conflicts involve concurrent *changeAttribute* operations where they are changing the same attribute of the same object to different values (such operations are *non-transformable* and the effects cannot be realised together).

```
void forward_transform( $op_1, op_2$ ) {
  if ( $op_1$  is a delete op OR a changeAttr op) return  $op_1$ ;
  if ( $op_2$  is a changeAttr op) return  $op_1$ ;
  if ( $op_2$  is a delete operation) {
    if ( $pos_1 > pos_2$ ) {  $pos_1 = pos_1 - 1$ ; }
    return  $op_1$ ;
  }
  if ( $op_2$  is an insert operation) {
    if ( $pos_1 > pos_2$ ) {  $pos_1 = pos_1 + 1$ ; return  $op_1$ ; }
    if ( $pos_1 < pos_2$ ) return  $op_1$ ;
    if ( $pos_1 = pos_2$ ) {
      if ( $char_1 = char_2$ ) { return  $op_1 = dup(op_2)$ ; }
      if ( $t_{op_1} > t_{op_2}$ ) return  $op_1$ 
      else {
         $pos_1 = pos_1 + 1$ ;
        return  $op_1$ ;
      }
    }
  }
}
```

Figure 1: Forward transformation function

Transforming an *insert* or *delete* operation against a *changeAttribute* operation or vice versa will result in the same operation since they do not affect each other. The transformation function is basically similar to the forward transformation (or inclusive transformation) introduced in previous papers (Sun, Jia, Zhang, Yang and Chen 1998,

Ellis and Gibbs 1989, Suleiman, Cart and Ferrié 1998). However, since each character is identified by an object id, every transformation function that involves the *delete* operation is slightly different. Figure 1 outlines the forward transformation function. Note that the transformation function is dependent on the application semantics and the operation primitives used in the application.

4 Proposed Algorithm

4.1 Overview

This section proposes a conflict management algorithm that combines operational transformation and multi-versioning to handle exclusive and non-exclusive conflict, while respecting user intention at the semantic level. In general, the algorithm is implemented as follows.

Firstly, for non-exclusive conflicts, operational transformation can be used to transform one of the conflicting operations against the other to preserve both intentions. Users can then choose to keep all operations or to undo some operations or to generate operations to fix any error created by the conflict. The consistency management scheme proposed by Citro, McGovern, and Ryan (2005) is based on operational transformation and as such is used as the basis for the algorithm described in this section. Not only does it ensure consistency in the presence of transformable operations, it also minimises resource usage, making it suitable for use in mobile environments. For specifics of the original algorithm, readers can refer to (Citro, McGovern, and Ryan 2005).

Secondly, the proposed algorithm implements a variation of post-locking called *delayed post-locking*, which addresses the partial-intention problem described in section 2.4. In general, whenever object versions are created, whether it is due to an exclusive or non-exclusive conflict, the object in conflict must be locked to avoid further update and to trigger conflict resolution. A post-lock can be placed either at the object level or at the object's attribute level depending on the application. If an application uses the attribute of an object as the base of the conflict (target), it may choose to lock the attribute of the object rather than the whole object, leaving the other attributes editable by the users. The finer the granularity of the lock, the higher the level of the concurrency supported, however the less likely a conflict will be noticed, thereby increasing the potential difficulty of resolving it. The specifics of delayed post locking, which are unique to the newly proposed algorithm, are described in section 4.2.

Thirdly, since exclusive conflicts cannot be resolved using operational transformation, a multi-versioning approach, in which each user's intention is realised in a different version of the document object, is implemented. Note that the application may, however, choose to use a multi-versioning approach to handle some non-exclusive conflicts as shown in the following example. Suppose *Alice*, *Bob* and *Cameron* are collaborating and they are concurrently modifying the word "and". *Alice* is trying to create the word "strand", *Bob* is making the word "grand", and *Cameron* is changing the word into

"errand". Using operational transformation approach, these operations can be transformed and executed at all sites without having to create object versions. However, depending on the current site states and operation timestamps, the new word would contain all the changes and become something like "sgettrand" or "stgerrand". The conflict is technically non-exclusive as the conflicting operations can be realised together consistently in the same object using operation transformation approach. However, *Alice* would not be able to determine what *Bob* and *Cameron* are trying to do. *Alice* would not have any idea if *Bob* is trying to make a word "grand". Furthermore, she would not know whether *Bob* is typing his letters concurrently with or after hers. Therefore, in this scenario, the conflict is better treated as an exclusive one and user intentions are better preserved using the multi-versioning approach. The specifics of the multi-versioning approach used in the present algorithm are described in detail in section 4.3.

Finally, the novel use of a *conflict table* to store all conflict information for the purpose of facilitating conflict resolution is also described in section 4.3 as part of the description of the general scheme for managing the storage and resolution of conflicts in order to support different conflict *resolution* strategies, with section 4.4 presenting one such strategy that is suitable for mobile environments.

4.2 User Intention Completion

A conflict can only be resolved properly when the intentions of all users involved in the conflict are completely realised (ie. the users have finished generating all necessary operations to the object and the other users have received all of them).

Suppose *Alice* in site S_1 and *Bob* in site S_2 , concurrently updating an object X (Figure 2(a)). To realise her intention, *Alice* needs to execute (generate) three operations on X : op_1 , op_2 and op_3 sequentially. Concurrently, *Bob* generates operations op_4 and op_5 to realise his own intention on X .

Using conventional post-locking approach (Xue, Zhang and Sun 2000), when op_4 , which conflicts with op_1 , arrives at S_1 before *Alice* generates op_2 , X is automatically locked, therefore *Alice* cannot fully realise her intention (Figure 2(b)). Consequently, *Bob* will not have the complete picture of *Alice*'s full intention. The conflict could be better resolved semantically if *Alice*'s intention is fully realised. This also happens if op_1 arrives at S_2 before op_5 is generated and op_1 is conflicting with op_5 .

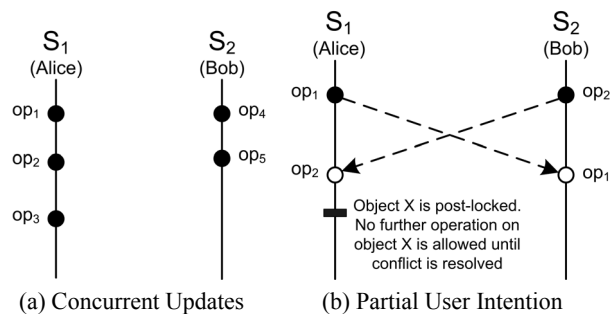


Figure 2: Examples of Collaboration Scenarios

Consequently, a strategy called *delayed post-locking* is proposed as the solution to this problem. This strategy allows users to completely realise their intention without being interrupted by incoming conflicting operations. The delayed post-locking technique uses a lock called a *user intention lock (UIL)* to prevent any interruption from incoming operations, thereby allowing the user to fully realise his/her intention. The rest of this subsection proposes a number of alternatives for implementing user intention locks.

4.2.1 Manual UIL

Each user is able to apply a UI-lock to his/her document when s/he wants to generate more than one operation to realise an intention. When a UIL is placed on a certain object, all incoming remote operations that target the UI-locked object are held in the remote operation queue even though they are causally ready. When the UIL is released, the operations waiting in the remote queue can then be executed accordingly.

Using the above example, *Alice* UI-locks *X* and generates op_1 , op_2 and op_3 . Although op_4 arrives before op_2 is generated, it cannot be executed because *X* is UI-locked (Figure 3). This gives *Alice* time to generate necessary operations, op_2 and op_3 , before being interrupted by incoming remote operations (and potentially post-locked due to conflict). The operations generated during the application of the UIL are technically concurrent with the incoming remote operations even though op_4 arrives at S_1 before op_2 is generated. When op_1 arrives at S_2 , *Bob* knows that there are more than one operation concurrent and conflicting with op_4 , therefore the conflict can be resolved after all operations have been received and it can be better resolved since *Bob* will know the full intention of *Alice*. A UIL is a local and temporary lock, that when *Alice* places a UIL on object *X*, *Bob* does not know anything about it and *Bob* only knows that op_1 and op_2 are concurrent with op_4 .

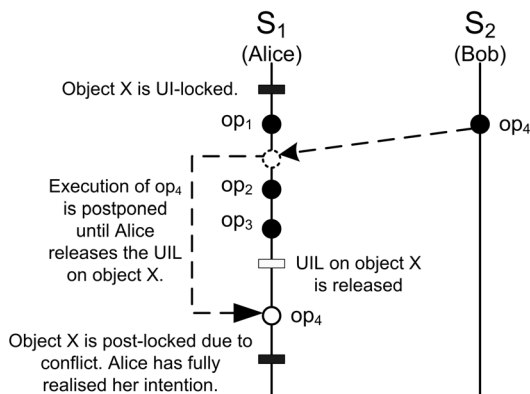


Figure 3: Manual UIL

4.2.2 Automatic UIL

The manual UIL described above requires manual intervention from the user. In practice, the application of the UIL can be automated. One option is to automatically place the UIL on an object whenever there is an incoming conflicting operation that targets the same object. Using the previous example, object *X* is UI-locked automatically when op_4 arrives and op_4 is held in the remote queue

(Figure 4). *Alice* is then notified that there is a conflicting incoming operation in the remote queue and *Alice* is given chance to complete any necessary operations on object *X* before the object is post-locked due to conflict. After generating op_2 and op_3 , *Alice* can then release the UIL to allow the remote operations to be executed as usual. A timeout period can also be applied to the UIL that when the timeout has elapsed and the user does not do anything to the object, the UIL is automatically released so as not to hold up the remote operations in the queue. Without losing generalities, this scheme of automatic UIL is used for the rest of the paper. Another possible option is to automatically place a UIL on the object currently being modified. Any conflicting remote operation targeting the object is held up in the queue. Once the user modifies another object, the UIL on the object is released and the remote operation can be released from the queue.

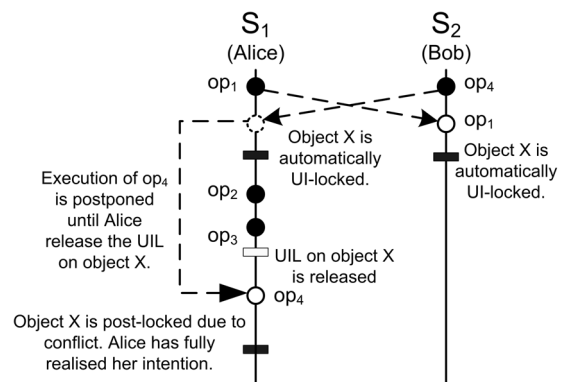


Figure 4: Automatic UIL

Nevertheless, regardless of when the UIL is placed and released, this strategy ensures that when a conflict on an object arises, the user has already generated all necessary operations on that object. The only drawback to this strategy is that it decreases the level of concurrency, and there is a trade-off between the level of concurrency support and the resolution of the partial-intention problem.

This strategy can also be extended to allow a user to inspect at the incoming operations before generating additional operations. When *X* is locally-locked automatically and op_4 is in the remote queue, *Alice* can choose to have op_4 executed to see what op_4 is trying to do. If *Alice* is happy with op_4 and does not want to generate further operations, *Alice* can undo op_1 and release the UIL, and op_4 is taken out from the remote queue. Otherwise, *Alice* can undo op_4 and store op_4 back to the remote queue and then generates op_2 and op_3 before the local lock is released. This gives flexibility on the user to either realise his/her intention or pre-empt his/her intention knowing that the intention of the other user is more desirable.

4.3 Conflict Management

This section describes the general data structures and techniques required to manage the storage and resolution of conflicts, in order to support different conflict resolution strategies, as described in the following section.

4.3.1 Participants

Each site maintains a list of participants (PL_S = participant list of site S) that stores the ids of the other sites as well as other information, such as site name and site connectivity information (IP address and port number).

4.3.2 State Vector

Besides its own state vector, each site maintains a State Vector Table that consists of N state vectors, one per site. Let SVT_k be the state vector table at site S_k . Initially, $SVT_k[i][j]=0$, $\forall i,j \in \{0, \dots, N-1\}$. After executing an operation op from a remote site S_r , timestamped by SV_{op} , the $SVT_k[r]$ is updated as follows: $SVT_k[r][i] = SV_{op}[i]$, $\forall i \in \{0, \dots, N-1\}$. Obviously, $SVT_k[i]$ at site S_k represents the view (or knowledge) of site S_k about the state at site S_i , $\forall i \in \{0, \dots, N-1\}$. In particular, $SVT_k[k]=SV_k$, i.e., $SVT_k[k]$ is the local state vector.

Each site also maintains a Minimum State Vector (MSV). MSV_k reflects the knowledge of site S_k about the number of operations which have been executed at every site. Initially $MSV_k[j] = 0$, $\forall j \in \{0, \dots, N-1\}$. After executing an operation and updating other elements of the SVT_k , site S_k updates $MSV_k[j]$ as follows: $MSV_k[j] = \min(SVT_k[i][j])$, $\forall i,j \in \{0, \dots, N-1\}$. If the value $MSV_k[i] = m$, then the first m operations generated at site S_i must have been executed at all sites.

4.3.3 Conflict Table

Whenever conflicts arise, existing multi-versioning schemes (Sun and Chen 2002, Chen and Sun 1999, Xue, Zhang, and Sun 2001, Xue, Orgun and Zhang 2003) either create complete alternative document versions, or store the conflicting object versions as well as the conflicting operations. In the proposed algorithm, whenever a conflict occurs, the object in conflict is locked, the user is notified (e.g. by highlighting the object), and only the operations causing the conflict are stored in a conflict table. This reduces memory consumption compared with existing solutions because the object versions can either be inferred by looking at the operations involved, or explicitly by executing the operations on a temporary document copy which can be destroyed following the inspection.

Each site maintains a Conflict Table (CT) to store the conflict information. Each entry in CT_k (Conflict Table of site S_k) is a tuple representing the conflicting object version: $CT_k[i] = \langle target, siteId, opIds, status, res \rangle$. $CT_k[i][target]$ is the target object on which the conflict occurs. Two operations are conflicting if they have the same target but modify it to different values. Depending on the definition of the conflict (application specific), $CT_k[i][target]$ can be the object id or the combination of the object id and object attribute. For example, $CT_k[i][target] = \langle X, FontSize \rangle$ if two or more operations are modifying the font size of object X to different values. Based on the previously stated assumption, this paper defines:

1. $CT_k[i][target] = \langle objId, attr \rangle$ if the operations are modifying the same object's attribute, and

2. $CT_k[i][target] = \langle objId \rangle$ if the operations are inserting characters at the same position of the same word.

$CT_k[i][objId]$ is the id of the target object and $CT_k[i][attr]$ is the attribute of $objId$ being modified.

$CT_k[i][siteId]$, $CT_k[i][opIds]$, $CT_k[i][state]$, and $CT_k[i][res]$ are the site id, operation ids, state and the resolution of $CT_k[i]$ respectively.

The following definitions are used to help in explaining the proposed algorithm.

Definition 3. Conflict-related Relation

$CT_k[i]$ and $CT_k[j]$ are *conflict-related*, $CT_k[i] \otimes_{ct} CT_k[j]$ iff $CT_k[i][target] = CT_k[j][target]$.

Definition 4. Site's involvement in a conflict

Site S_k is involved in $CT_k[i]$, $S_k \in_{ct}^S CT_k[i]$ iff there exists $CT_k[j]$ such that $CT_k[j] \otimes_{ct} CT_k[i]$ and $CT_k[j][siteId] = S_k$.

Definition 5. Operation's involvement in a conflict

Operation op_x is involved in $CT_k[i]$, $op_x \in_{ct}^{op} CT_k[i]$ iff there exists $CT_k[j]$ such that $CT_k[j] \otimes_{ct} CT_k[i]$ and $op_x \in CT_k[j][opIds]$.

$CT_k[i][status]$ signifies whether or not the intention on the particular object version has been completely realised. This information allows users to make better conflict resolution decisions in terms of which document version should be accepted. $CT_k[i][status]$ be one of the following values:

- *Partial*. $CT_k[i][status] = partial$ if it is in neither complete nor resolvable. This signifies that this entry is recently created, and the user of site $CT_k[i][siteId]$ might not have generated all necessary operations to fully realised his/her intention (partial intention problem).
- *Complete*. $CT_k[i][status] = complete$ if site $CT_k[i][siteId]$ has finished generating conflicting operations. In other words, site $CT_k[i][siteId]$ has already executed at least one of the conflicting operations ($\exists op_x \in \{CT_k[j][opIds], \forall CT_k[j] \otimes_{ct} CT_k[i] \text{ and } CT_k[j][siteId] \neq CT_k[i][siteId]\}$). Suppose $S_j = CT_k[i][siteId]$, as soon as S_j executes one of the conflicting operations, the target object in S_j is locked, therefore S_j cannot generate further operations on the object, thus S_j has finished generating conflicting operations.
- *Resolvable*. $CT_k[i][status] = resolvable$ iff site S_k has the right to resolve the conflict (ie. potentially accept $CT_k[i]$). Although a conflict can be resolved by various conflict resolution strategies (as mentioned in section 2.5), a conflict is eventually resolved by one mobile site S_k selecting one of the Conflict Table entries $CT_k[i]$ to be the final version for that particular object. This decision is then broadcast to all other sites. Note that the criteria to determine whether any particular site should be allowed to choose a solution (i.e. $CT_k[i][status] = resolvable$) is referred to in this paper as a *conflict resolution strategy*, with one such strategy is presented in section 4.4.

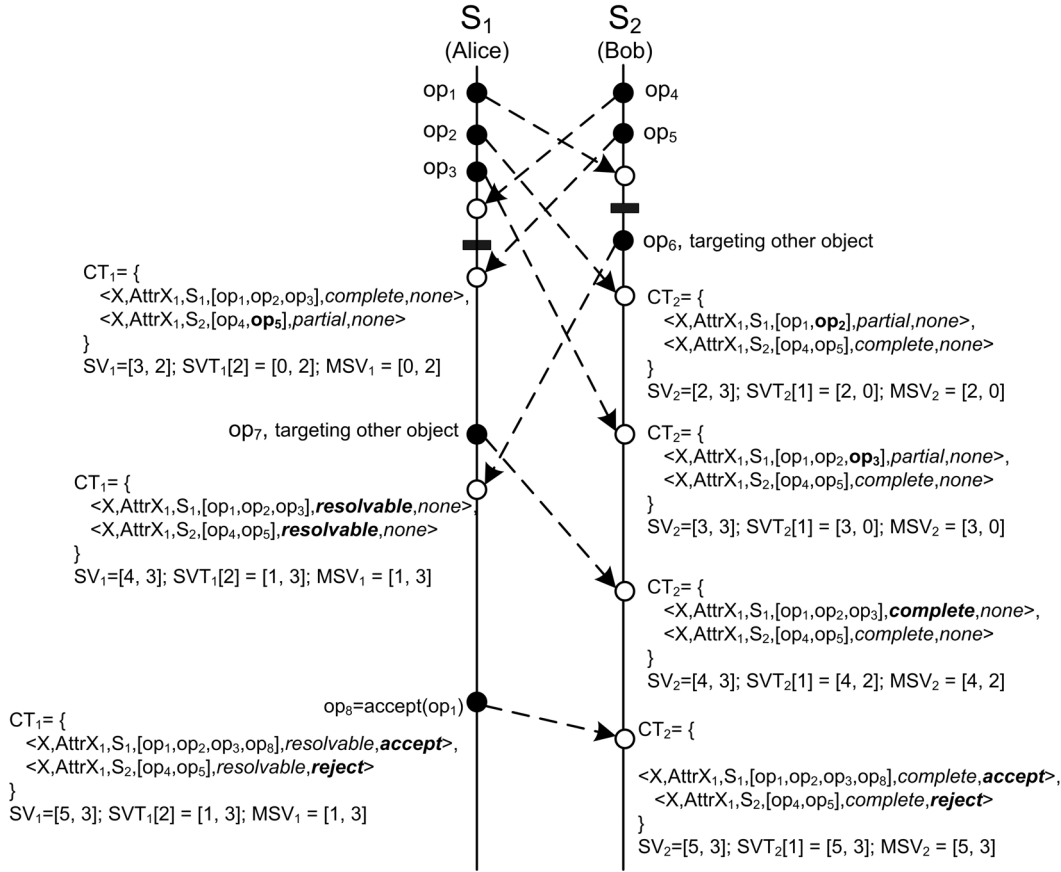


Figure 5: Conflict table entries

$CT_k[i][res]$ denotes whether $CT_k[i]$ is accepted as the final version. $CT_k[i][res] = accept$ if $CT_k[i]$ is accepted as the final version, $reject$ if it is rejected or $none$ if the conflict has not been resolved.

From the above example (Figure 3 and 4), when op_4 is executed at site S_1 , two entries in CT_1 is created:

1. $CT_1[0] = \langle X, S_1, [op_1, op_2, op_3], complete, none \rangle$, and
2. $CT_1[1] = \langle X, S_2, [op_4], partial, none \rangle$.

When a conflicting remote operation arrives from a site, and there is already a CT entry of that site, the operation is simply appended to the $opIds$ for that CT entry. For example, when op_5 arrives and is executed in site S_1 , op_5 is simply added to $CT_1[1]$ to become $\langle X, S_2, [op_4, op_5], partial, none \rangle$. Both entries are conflict-related since they are involved in the same conflict (ie. they have the same target). Site S_1 and S_2 are involved in $CT_1[0]$ and $CT_1[1]$ because they generate the conflicting operations. Operations op_1, op_2, op_3, op_4 and op_5 are involved in $CT_1[0]$ and $CT_1[1]$ as they are the conflicting operations.

Figure 5 illustrates the creation and modifications of CT entries when the sites receive each of the conflicting operations. Note that as soon as S_2 receives op_7 , S_2 realises that S_1 has finished generating the operation on object X based on the state vector of op_7 , therefore the status of the respective conflict table entry CT_2 is changed to *complete*.

Figure 6 outlines the *check_conflict(op)* procedure that checks whether or not an incoming operation op causes or is involved in a conflict in site S , and add the op into a CT entry accordingly. Note that without losing generality,

automatic UIL is being used as an example in this procedure.

```

void check_conflict(op) {
  for all  $op_i$  such that  $op_i || op$  and  $op_i \in HB_S$  {
    if ( $op_i$  conflicts with  $op$ ) {
      UI-lock( $target_{op}$ );
      wait until UIL( $target_{op}$ ) is released;

      if there exists  $CT_S[i]$  in  $CT_S$  such that
      ( $CT_S[i][target] = target_{op}$  AND  $CT_S[i][siteId] = S_{op}$ ) {
         $CT_S[i][opIds] += op$ ;
      } else {
         $CT_S += \langle target_{op}, S_{op}, [op], partial, none \rangle$ ;
      }

      if there exists  $CT_S[j]$  in  $CT_S$  such that
      ( $CT_S[j][target] = target_{op}$  AND  $CT_S[j][siteId] = S$ ) {
         $CT_S[j][opIds] += op$ ;
      } else {
         $CT_S += \langle target_{op}, S, [op_i], complete, none \rangle$ ;
      }
    }
  }
}

```

Figure 6: Conflict checking procedure

As previously stated, a conflict is eventually resolved by selecting one Conflict Table entry to be the final version for that particular conflict. If $CT_k[i]$ is selected to be the desired version, a Conflict Resolution Operation (CRO), $op_r = accept(op_x)$, is generated where op_r is one of the operation in $CT_k[i][opIds]$. If $CT_k[i]$ is accepted, then $CT_k[i][res] = accept$, and the CT entries that are conflict-related to $CT_k[i]$ are rejected ($CT_k[j][res] = reject, \forall$

$CT_k[j] \otimes_{ct} CT_k[i]$). The *CRO* is then broadcast to all other sites and when it arrives at a site, say site S_j , site S_j will accept the Conflict Table entry which op_x belongs to, and reject the other conflict-related Conflict Table entries.

Figure 5 illustrates the conflict resolution process and the changes made to the conflict table entries. Note that while the proposed conflict management strategy can be used with various conflict resolution strategies, figure 5 illustrates one example of a possible conflict resolution strategy as discussed in section 4.4. The *CRO* op_r is then appended to $CT_k[i][opIds]$ for garbage-collection purpose. Since the conflicting operations are non-transformable, all operations that in conflict to $CT_k[i][opIds]$ are simply undone (if they have been executed) and operations $CT_k[i][opIds]$ are executed and stored in the history replacing the undone operations.

4.3.4 Conflict Table Garbage Collector

An entry in $CT_k[i]$ needs to be kept in CT_k until site S_k is confident that the conflict has been resolved and the entry is not needed anymore to process future operations. If $CT_k[i]$ is rejected, then site S_k can remove $CT_k[i]$ if S_k is confident that there will not be any future conflicting operation that comes from site $CT_k[i][siteId]$ (ie. $CT_k[i][status]$ is *complete* or *resolvable*). On the other hand, if $CT_k[i]$ is accepted, $CT_k[i]$ can only be removed if all sites have executed all operation in $CT_k[i][opIds]$ including the *CRO* (*CRO* is appended to $CT_k[i][opIds]$ once it is accepted), which also means that all sites have received results of the conflict resolution. In summary, $CT_k[i]$ can be removed from CT_k iff:

- $(CT_k[i][res] = reject) \text{ AND } ((CT_k[i][status] = complete) \text{ OR } (CT_k[i][status] = resolvable))$, OR
- $(CT_k[i][res] = accept) \text{ AND } (MSV[S_{op_i}] > SV_{op_i}[S_{op_i}], \forall op_x \in CT_k[i][opIds])$.

4.4 Conflict Resolution

Having described in the previous section a general conflict management mechanism, this section presents an example of a specific conflict resolution strategy. Compared to a voting strategy (consensus reaching strategy), the conflict resolution strategy presented in this section uses less resources as it requires less message roundtrips and does not require a complicated consensus reaching algorithm. As such, it is suitable for mobile network environments. In addition, conflict is resolved without waiting for all sites to receive the conflicting operations thereby increasing user responsiveness. However, despite these advantages the algorithm does not facilitate negotiation or the reaching of consensus since this is beyond the scope of this paper and thus left to future work.

Under this strategy, each site has a conflict resolution priority level (PR_{S_k} = priority level of site S_k), which is ordinal relative to other sites. The sites' priorities can be static (e.g. based on *siteId*) or dynamic (e.g. based on which site generated the conflicting operation the earliest). To avoid relying on a single site for conflict resolution (usually the site with the highest priority), only the priority level of the sites actually involved in a given conflict are considered. In other words, the user in site S_k

has the right to resolve conflict if site S_k is involved in the conflict and site S_k has the highest priority among all sites involved in the conflict. If there is another site that has a higher priority than S_k , say S_r , site S_k still has the right to resolve the conflict if S_k is sure that S_r is not involved in the conflict.

Therefore, more formally, site S_k has the right to resolve conflict (generate $op_r = accept(op_x)$, where $op_x \in CT_k[i][opIds]$) iff:

- S_k is involved in the conflict ($S_k \in_{ct} CT_k[i]$), and
- PR_{S_k} is the highest among all sites involved in the conflict ($PR_{S_k} \geq PR_{S_x}, S_x \in \{CT_k[j][siteId], \forall CT_k[j] \otimes_{ct} CT_k[i]\}$), and
- Sites with higher priority (if any) are not involved in the conflict. In other words, sites with higher priority have executed at least one of the operations involved in the conflict. For all $S_x \in PL_k$ and $PR_{S_x} > PR_{S_k}$, there exists op_j such that $SV_{op_j}[S_{op_j}] > SV_{op_j}[S_{op_j}]$ and $op_j \in_{ct} CT_k[i]$.

When $CT_k[i][status] = complete$ and the user in site S_k has the right to resolve the conflict, the status of CT_k becomes $CT_k[i][status] = resolvable$. As described in the previous section, once $CT_k[i]$ has been accepted, a *CRO* is generated and broadcast to all other sites.

The conflict resolution strategy described above has the following advantages. Firstly, when a site has the right to resolve conflict, it can resolve the conflict anytime without having to wait until all sites receive all conflicting operations. Secondly, sites that are not involved in the conflict do not need to resolve conflict; therefore it does not always have to be a pre-determined or delegated group member who resolves conflicts. Thirdly, the rightful site can resolve conflict anytime without having to wait for the lock to be synchronized. A site whose priority is not the highest still has to make sure that the higher-priority sites are not involved in the conflict. This can be determined by the state vector of the operations generated by those sites (note that each operation carries the state vector of the originator site at the time it is generated). Therefore, it has to wait for operations to be generated by higher-priority sites before it can conclude that it has the right to resolve conflict. This is, however, still better than having to wait for all sites to receive all conflicting operations, and this process can be expedited by sending a state vector request to higher-priority sites so they can send their state vector update without having to generate operations.

5 Conclusion

The algorithm presented in this paper has built upon existing techniques such as operational transformation and multi-versioning to more effectively handle conflicts in replicated object based collaboration applications. In particular the concepts of delayed post-locking and conflict tables have been introduced to address shortcomings in existing approaches.

Delayed post-locking uses *user intention locks* (UIL) to locally lock the object being edited so that while the UIL is in place, all incoming operations that target that object will be held in the queue until the UIL is released. This

gives time for the user to generate necessary operations to fully realise his/her intention on the object so that more complete information is available to other users to assist in the conflict management process.

Furthermore, each site maintains a *conflict table*, which facilitates the user in dealing with conflict by letting him or her know whether other users have fully realised their intention on the object in conflict. It also informs the user when s/he has the right to resolve the conflict, insofar as being allowed to select the final version of the object.

Most importantly, the proposed algorithm satisfies the following conditions: it does not suffer from a partial-intention problem; it need not depend on a group leader or other pre-specified conflict resolution roles; the conflict can potentially be resolved without having to wait for all sites to receive all conflicting operations (dependent upon the chosen conflict resolution strategy); and finally, the algorithm provides better information to users so that they can resolve the conflict knowing the status of the conflict.

Currently a prototype of the algorithm has been implemented and tested in a simulation environment. Future work will be to complete the full implementation of the algorithm and its integration with consistency management, membership management and document partitioning algorithms as part of a larger project aimed at providing a comprehensive framework for real time mobile collaboration. Future work will also look at alternative conflict resolution strategies, with particular emphasis on their effectiveness from a usability point of view, and their performance and impact on resource consumption within a mobile environment.

6 References

- Xue, L., Zhang, K., and Sun, C. (2001): An integrated post-locking, multi-versioning, and transformation scheme for consistency maintenance in real-time group editors. *Proc. ISADS*, 57–64
- Citro, S., McGovern, J. and Ryan, C. (2005): An efficient consistency management algorithm for real-time mobile collaboration. *Proc. International Conference on Quality Software*, Melbourne, Australia, 287-294, IEEE Computer Society.
- Dewan, P. (1999): Architectures for collaborative applications. In *Computer supported Cooperative Work*. 169-193. Beaudouin-Lafon, M., (ed). John Wiley & Sons.
- Sun, C. and Chen, D. (2002): Consistency maintenance in real-time collaborative graphics editing systems. *ACM Transactions on Computer-Human Interaction*, **9**(1):1-41.
- Munson, J. and Dewan, P. (1996): A concurrency control framework for collaborative systems. *Proc. ACM Conference on Computer Supported Cooperative Work*, 278-287, ACM Press.
- McGuffin, L.J. and Olson, G.M. (1992): Shredit: a shared electronic workspace. *CSMIL Technical Report 45*, The University of Michigan.
- Newman-Wolfe, R.E., Webb, M.L. and Montes, M. (1992): Implicit locking in the ensemble concurrent object-oriented graphics editor. *Proc. ACM Conference on Computer Supported Cooperative Work*, 265-272, ACM Press.
- Ellis, C.A. and Gibbs, S.J. (1989): Concurrency control in groupware systems. *Proc. ACM SIGMOD International Conference on Management of Data*, 399-407, ACM Press.
- Suleiman, M., Cart, M. and Ferrié, J. (1998): Concurrent operations in a distributed and mobile collaborative environment. *Proc. IEEE International Conference on Data Engineering (IEEE / ICDE '98)*.
- Vidot, N., Cart, M., and Ferrié, J. and Suleiman, M. (2000): Copies convergence in a distributed real-time collaborative environment. *Proc. ACM Conference on Computer Supported Cooperative Work*, 171-180, ACM Press.
- Sun, C., Jia, X., Zhang, Y., Yang, Y. and Chen, D. (1998): Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Transaction on Computer-Human Interaction* **5**(1):63-108.
- Chen, D. and Sun, C. (1999): A distributed algorithm for graphic objects replication in real-time group editors. *Proc. International ACM SIGGROUP Conference on Supporting Group Work*, 121-130, ACM Press.
- Xue, L., Zhang, K. and Sun, C. (2000): Conflict control locking in distributed cooperative graphics editors. *Proc. First International Conference on Web Information Systems Engineering*, **1**: 401-408, IEEE Computer Society.
- Sun, C. (2002): Optional and responsive fine-grain locking in internet-based collaborative systems. *IEEE Transactions on Parallel and Distributed Systems* **13**(9): 994-1008.
- Sun, C. and Sosič, R (1999): Optional locking integrated with operational transformation in distributed real-time group editors. *Proc. Annual ACM Symposium on Principles of Distributed Computing*, 43-52, ACM Press.
- Xue, L., Orgun, M. and Zhang, K. (2003): A multi-versioning algorithm for intention preservation in distributed real-time group editors. *Proc. Australasian Conference on Computer Science: Research and Practice in Information Technology*, 19-28, Australian Computer Society, Inc.
- Chu-Carroll, J. and Carberry, S. (1995): Response generation in collaborative negotiation. *Proc. Annual Meeting on Association for Computational Linguistics*, Morristown, NJ, USA, 136-143, Association for Computational Linguistics.
- Fischer, M.J., Lynch, N.A. and Paterson, M.S. (1985): Impossibility of distributed consensus with one faulty process. *Journal of ACM* **32**(2):374-382.
- Coulouris, G., Dollimore, J. and Kindberg, T. (2001): Distributed Systems Concepts and Design. Addison Wesley.

Architecture of a Web Accelerator for Wireless Networks

Jian Song, Yanchun Zhang

School of Computer Science and Mathematics, Victoria University
Melbourne, Victoria 8001 Australia,

jians@csm.vu.edu.au, yanchun.zhang@vu.edu.au

Abstract

With the continuous growth of mobile users and web-based wireless applications, the performance of accessing web services via wireless is becoming one of the key issues. Regular TCP has many problems to tackle wireless connections. Especially for 2.5G or 3G mobile users, the bandwidth is always limited to access various IP applications. In this paper, we describe the design, implementation and evaluation of a web accelerator. Our proposed web accelerator increases the web pages retrieval speed and maximizes bandwidth utilization of TCP/IP application to overcome the obstacles of wireless networks. This goal has been achieved by two solutions: first, replacing TCP with *Boosted TCP Protocol* (BTCP) to improve the performance of TCP over wireless; second, compressing the web content to make it efficient for various wireless devices.

Keywords: TCP (Transmission Control Protocol), Web acceleration, Wireless Network, Compression.

1 Introduction

As mobile users and web-based wireless applications continue to grow, the wireless accesses to a variety of web services through various devices such as laptop, PCs, PDAs, smart phones, etc., are becoming very popular today. However, as applications shift from the traditional transaction exchange to Internet/Intranet access, TCP and its performance over wireless links are not successful as wired networks. Through researches and experiments, it is apparent that TCP over wireless links results in many problems (George Xylomenos, G.C. Polyzos, Petri Mahonen and Mika Saaranen 2001).

Characteristics of wireless link have significant effects on TCP performance, including latency of physical layer, bandwidth oscillation, asymmetric uplink and downlink, delay spikes and intersystem handovers etc. Many features of wireless networks can result in various delays e.g. (H. Inamura, G. Montenegro, R. Ludwig, A. Gurtov, and F. Khafizov 2003).

One example in 2.5G/3G networks, a *delay spike* is an abrupt increase in the latency of the communication path. 2.5G/3G links are likely to experience *delay spikes* exceeding the typical *round-trip-time* (RTT) by several

times. It can cause spurious TCP timeouts, unnecessary retransmissions and a multiplicative decrease in the congestion window size e.g. (F. Xin, A. Jamalipour 2005).

Another example, in mobile networks, if multiple users want to transfer large amounts of data at the same time, the scheduler may have to repeatedly allocate and de-allocate resources for each user. Periodic allocation and release of high-speed channels can be referred to as *Bandwidth Oscillation*. *Bandwidth Oscillation* effects such as spurious retransmissions were identified e.g. (M. Scharf, M. Necker, and B. Gloss 2004 and Ludwig, R. and R. H. Katz 2000), as factors that degrade throughput. There are research studies such as (Khafizov, F. and M. Yavuz 2002 and Yavuz, M. and F. Khafizov 2002), which show that in some cases *Bandwidth Oscillation* can be the single most important factor in reducing throughput.

Many modifications of TCP have been pointed out to eliminate the degrading of TCP performance over wireless networks, such as M-TCP (K. Brown and S. Singh 1997), I-TCP (Ajay Bakre, B.R.1995), W-TCP (P. Sinha, N. Venkitaraman, R. Sivakumar, V. Bharghavan 1999), snoop and Freeze-TCP (Goff T, Moronski J, Phatak DS. 2000). Whilst, narrow bandwidth, slow response and high resource utilization are main difficulties for wireless network users.

In this paper, we propose a web accelerator which combines TCP optimization and web content compression together to eliminate these obstacles. Firstly, we present a solution to address the limits of TCP using *Boosted TCP* (BTCP). We replace the TCP protocol with the BTCP to enhance the transport layer performance in wireless environment. From the experiment results, wireless terminals could experience the advantage of optimized bandwidth conditions that increase five fold from the usual TCP connection. Then we discuss the web content compression strategy based on the zlib algorithm (P. Deutsch, J.-L. Gailly 1996) (same algorithm used in the popular internet tool gzip (L. Peter Deutsch 1996), which benefits the mobile clients when the available bandwidth is scarce (e.g. GPRS or 1-2Mbps wireless radio links). Compression gains depend on the content that is transferred.

The main contribution of this architecture is the combination of TCP optimization and web content compression in wireless environment, aiming to maximize the web surfing experience of wireless users, especially for 2.5G or low speed wireless connections. This platform has been tested in China Mobile GPRS network in Beijing.

The rest of paper is organized as follows. Section 2 presents the design of our web acceleration platform architecture. In the following two sections,

implementation and evaluation of this system are reported. We draw the conclusion and the future work in the last section.

2 Architecture of the Web Accelerator

Our web accelerator acts as a proxy server between clients and Internet. As shown in figure 1, the web accelerator could be placed in ISP (Internet Service Provider) side. Mobile users connect to Internet through BS (Base Station) by various wireless links (e.g. GSM, GPRS, CDMA, CDPD etc.). TCP optimization occurs between mobile clients and BS, which will be discussed in following section 2.2. Processing of the web compression is presented in section 2.3.

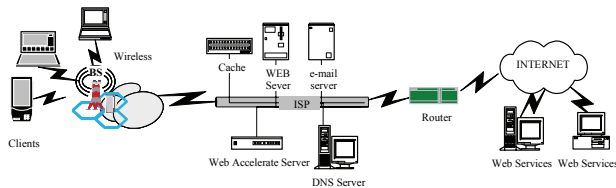


Figure 1: The Web Accelerator resides in ISP's local network and performs the TCP optimization and web content compression.

2.1 TCP over Wireless Links

TCP was specifically designed as a flexible protocol that can operate on various links such as dial-up to Gigabit Ethernet LANs, however it is not optimal on any of these links. Experience with TCP usage over wireless links proves that the performance and resource consumption of TCP over a wireless link is unacceptable (George Xylomenos, G.C. Polyzos, Petri Mahonen and Mika Saaranen 2001).

- *Slow Start.* TCP uses a congestion control algorithm to manage the volume data of the sender (W. Richard Stevens 1997). TCP performs a detection process called "Slow Start" whenever a connection is made because TCP is unaware of the link bandwidth. TCP calculates the lowest possible bandwidth and increases exponentially until a packet loss is detected. In the wireless environment, the long round trip time implies a long period during which the link is under-utilized and perceived by the user as slow.
- *Congestion Avoidance.* TCP assumes all packet loss is caused by congestion. Hence, there is a reduction in the current transmission rate by half. Also, the "slow start" algorithm is changed into a much slower acceleration. If the packet loss occurred due to link noise, or other transient conditions, TCP will still react in this manner. The result is unused link capacity which means a slower data transfer.
- *Receive Window Size.* An application that sends data using TCP is only allowed to send as many bytes that are explicitly allowed in the last packet seen from the intended destination. (This is to make sure the destination has enough space to store the bytes about to be sent.) This value is known as the "Advertised Receive Window." Typically, this is set to about 8 Kbytes, which severely limits the top speeds that a

TCP connection can transfer data over a wireless link. The actual bandwidth can be calculated according to the RTT and the window size using the formula (1):

$$\text{Max BPS} = \text{RWIN} / \text{RTT} \quad (1)$$

For example, consider a 1 Mbps wireless link with a 250ms delay and 4Kbytes window size -- only 128 Kbps will be sent on a 1 Mbps link. It will still limit TCP's speed.

- *3-way Handshake.* The TCP three-way handshake requires three RTTs before there is an initiation of sending data. It adds extra time for each wireless connection.

2.2 BTCP Basic Structure and Features

The BTCP is a TCP-like reliable protocol that replaces TCP over the path where performance optimization is needed. TCP packets from a source location are transformed from TCP to BTCP and sent to the destination. There they are transformed back into TCP and sent to the originally intended target.

Web Accelerator manages both BTCP and TCP flow control and data queues to allow a full protocol conversion. Protocol conversion is illustrated in figure 2.

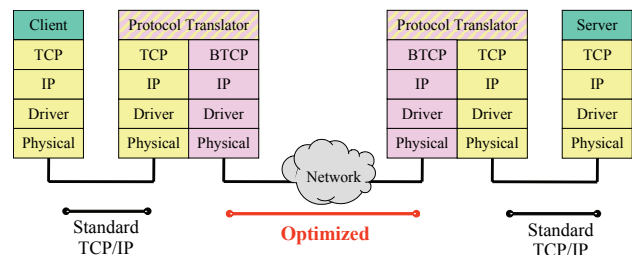


Figure 2: The Web Accelerator replaces TCP over the path of wireless links between mobile users and BS.

To address the TCP issues, our proposed BTCP improves the TCP performance by using following strategies.

- *Pre-Tuned Parameters.* BTCP consistently operates at a high data rate in order to maximize the use of bandwidth resources. It can be optimized to the specific link in use to take into account bandwidth, delay, bit error ratio, and Maximum Transmission Unit (MTU) size.
- *Immediate Transmission at Full Speed.* Unlike TCP, the BTCP protocol does not need to perform Slow Start in order to discover available bandwidth. Transmission begins at the full rated link speed from the first byte.
- *Changed Congestion Avoidance.* Unlike TCP, this slows data transfer dramatically after a lost packet is detected. BTCP always assumes lost packets are due to BER and not congestion. Thus, it will always recover from lost packets using immediate and selective retransmission without reducing speed or bandwidth utilization. As a result, BTCP can fill any wireless "pipe" regardless of latency or BER.
- *No 3-way Handshake.* The BTCP protocol keeps a tunnel open between the Client and the Server,

eliminating the need for the standard TCP 3-way handshake.

2.3 Web Content Compression

A compression system directly reduces the amount of traffic that needs to be transported, saves bandwidth, speeds up response time and shortens transfer times despite the CPU time required for decompressing the responses. Our proposed web accelerator offers lossless compression based on the zlib algorithm (same algorithm used in the popular internet tool gzip, and other compression algorithms may be used, too). Compression gains depend on the content that is transferred. Average overall compression ratios are about 25%-30% for Web traffic, which depends on what kind of contents.

The reason why we choose to use zlib is mainly because it is a good compression algorithm for which sources and specifications are easily available. Zlib supports compression level from 0 to 9, which means we could control the compression complexity and compression ratio between these 10 levels.

3 Implementation of the Web Accelerator

In order to evaluate and validate the web accelerator, we deployed our Solaris-based accelerator server in real ISP's local backbone network (China Mobile, Beijing). The environment of the implementation is illustrated as figure 3.

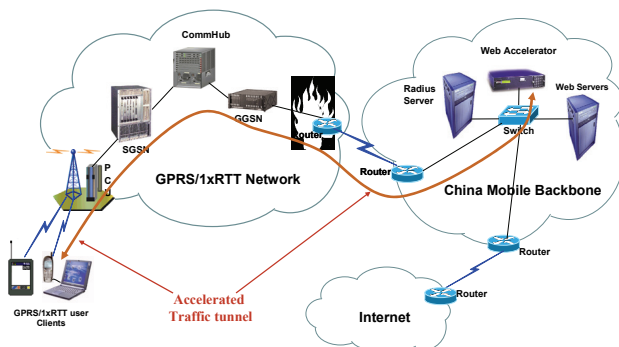


Figure 3: Implementation of Web Accelerator in ISP

The GPRS mobile clients get wireless link from the nearest BS, and get through the GPRS network to China Mobile Backbone, then connect to Internet. The mobile users could be client program installed or clientless mode. The whole accelerated traffic occurs between mobile users and our web accelerator.

The TCP traffic flow and HTTP connection over BTCP are shown in figure 4 and figure 5. From the client side, client listens on predefined TCP ports, each of which support one protocol (e.g. Http – 9090, IPA port – 9876, FTP – 9091, Free_TCP – xxxx, SSL – 9092). TCP Thread handles all TCP sockets from user applications to predefined ports. Then BTCP thread handles all out going traffic to accelerator.

In server side, Server receives new BTCP connections on predefined working port, which are acknowledged on new

socket from random port pool. TCP Thread handles all out going traffic from accelerator to the Internet.

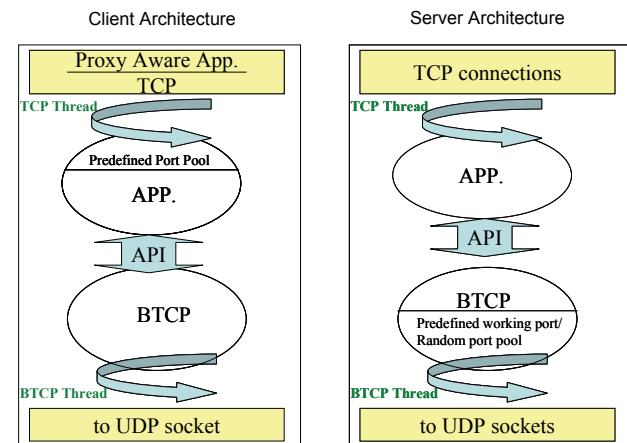


Figure 4: Client and Server architectures

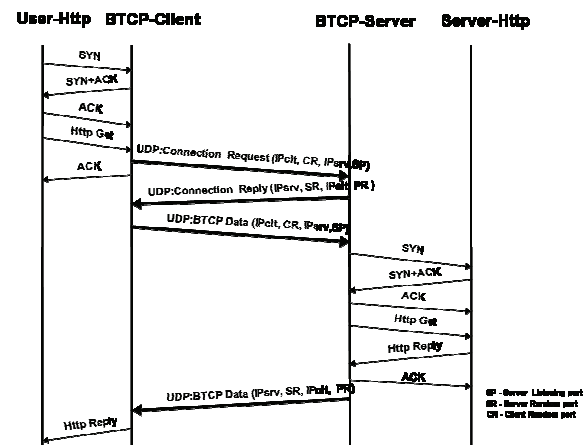


Figure 5: HTTP connections over BTCP

4 Evaluation of Performance Gain and User Experience

In this section, we have evaluated the performance of the web accelerator by three different set of experiments.

First, BTCP and TCP throughputs for various delays has been tested and compared. The following figure 6 shows a throughput comparison test for traffic transferred over GPRS link between TCP and BTCP. It is apparent that TCP throughput reduces significantly during long round trip time, which means that the link is under-utilized and perceived by the user as slow. On the contrary, BTCP only experiences a small decrease during same time.

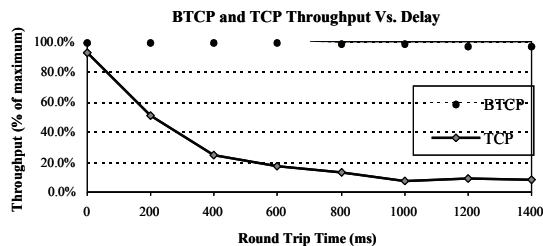


Fig. 6. BTCP throughput compared with TCP throughput for various delays

In the second experiment, we evaluated the HTTP performance over BTCP and TCP without compression. For each page retrievable with BTCP the browser starts with the unavoidable RTT as TCP for the first object. After that the user will receive constant data stream at high rate much like FTP transfer of the object. This results in faster page retrieval. Table 1 lists some web sites of test done on GPRS link.

URL	Download time (sec)	
	BTCP	TCP
http://www.cnn.com/WORLD/	7.4	17.4
http://www.goldmansachs.com/	6.3	14.9
http://www.fool.com/	6.4	16.3
http://cnfn.cnn.com/	5.3	13.1
http://www.amazon.com/	6.6	17.4

Table 1: Comparison of download time for BTCP and TCP

In the third experiment, we compared the web browsing of a few typical websites in three situations: standard TCP, BTCP and BTCP plus web content compression. Figure 7 visualizes the bandwidth savings achieved with different situations. We realized maximum savings through the reduction of TCP overhead required by ‘Slow Start’, congestion avoidance and web content compression. But, it will be noted that the savings in download bytes varied widely from site to site. The savings is based on the nature of the site. For example, the CNN (http://www.cnn.com/WORLD/) site is mostly text and therefore can be compressed very efficiently. The NASA (http://www.nasa.gov) site is mostly images that are already compressed (GIF or JPEG), therefore, fewer saving is realized.

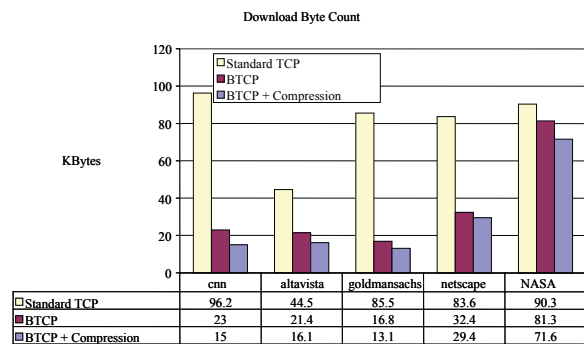


Fig. 7: Bandwidth savings of some websites with or without BTCP and compression

5 Conclusions and future work

In this paper, we presented the design, key points of the architecture, implementation and evaluation of a web accelerator. Our accelerator has been deployed and tested in a typical ISP's backbone to improve the performance of web browsing for GPRS users. As the results of experiments showed, mobile users benefited from our BTCP and content compression. We concluded that BTCP is useful to cope with the problems of TCP over wireless connections, and web content compression could optimize the utility of the bandwidth. Combining two solutions is a good way for wireless web browsing, but the performances differ from site to site, for they are based on the contents of the site. The text-based sites can be compressed more efficiently than picture-based sites.

In the current work, the lossy algorithm has not been incorporated in our accelerator, which can reduce the data size and the quality of compressed pictures (e.g. JPEG, GIF). In the future, we will continue the work with lossy compression. Another concern about our web accelerator is the additional CPU time and resource requirements for both client and server sides, which will decrease the performance under high demand situations.

6 References

- George Xylomenos, G.C. Polyzos, Petri Mahonen and Mika Saarinen (2001): TCP Performance Issues over Wireless Links, *IEEE Communications Magazine*.
- H. Inamura, G. Montenegro, R. Ludwig, A. Gurtov, and F. Khafizov (2003): TCP over second (2.5G) and third (3G) generation wireless networks. In RFC 3481.
- F. Xin, A. Jamalipour (2005): TCP performance in wireless networks with delay spike and different initial congestion window sizes. In *Computer Communications xx* (2005) 1-8
- M. Scharf, M. Necker, and B. Gloss (2004): The sensitivity of TCP to sudden delay variations in mobile networks. In *Springer LNCS series*, Proc. Networking 2004, Athens, Greece.
- Ludwig, R. and R. H. Katz (2000): The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions. *ACM Computer Communication Review* 30(1).

- Khafizov, F. and M. Yavuz (2002): Running TCP over IS-2000. Proc. of *IEEE ICC*.
- Yavuz, M. and F. Khafizov (2002): TCP over Wireless Links with Variable Bandwidth. Proc. of *IEEE Vehicular Technology Conference*.
- K. Brown and S. Singh (1997): M-TCP: TCP for Mobile Cellular Networks, ACM Computer Communications Review, vol27, no.5.
- Ajay Bakre, B.R. Badrinath (1995): I-TCP: Indirect TCP for Mobile Hosts, Tech Rep., Reuters university, <http://www.cs.rutgers.edu/badri/journal/contents11.htm> l.
- P. Sinha, N. Venkitaraman, R. Sivakumar, V. Bharghavan (1999): WTCP: a reliable transport protocol for wireless wide-area networks, *Proc. of ACM Mobicom '99*, Seattle, WA, pp. 231–241.
- Goff T, Moronski J, Phatak DS. (2000): Freeze-TCP: A true end-to-end TCP enhancement mechanism for mobile environments. In *Proceedings of the IEEE INFOCOM*, March.
- P. Deutsch, J.-L. Gailly (1996): ZLIB Compressed Data Format Specification version 3.3, RFC1950.
- L. Peter Deutsch, (1996): GZIP file format specification version 4.3, RFC1952.
- W. Richard Stevens (1997): TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms, IETF RFC 2001.

Periodical Payment Model using Restricted Proxy Certificates

Grigori Goldman

School of Information Technology and Electrical Engineering
University of New South Wales, Australian Defence Force Academy
Canberra 2600, Australian Capital Territory

grigori.goldman@adfa.edu.au

Abstract

In this paper we shall introduce a new electronic payment concept based on the popular direct debit payment model, entitled periodical payments. The direct debit model currently in use online is neither secure nor flexible, and requires a leap of faith by the customer who must trust the merchant to behave honestly. Electronic direct debit request (DDR) forms are not signed by both parties in a binding manner, which means that merchants can change the terms of DDR agreements post-fact. Unsigned DDR agreements give the merchant unprecedented power over customer accounts with little recourse for dispute.

In this paper we shall demonstrate how the use of restricted proxy certificates with cryptographic signatures can be adopted to support a new periodical payment model. A payment policy language is presented that is tailored towards specifying rules that govern precisely how and when merchants can access and transfer funds from customer accounts into their own. Using this model will ensure that mutually signed policies are instantly enforceable on every transaction within a payment period.

There is a fundamental difference between this proposal and other electronic payment schemes. Most such schemes attempt to replicate the features of physical cash such as anonymity, and therefore focus on single payment transactions that simulate cash changing hands. Since direct debit is a popular payment choice, our proposal provides significant improvement to this essentially paper-based payment model that currently does not integrate well in a purely electronic world.

Keywords: e-commerce, payment, periodical, direct debit.

1 Introduction

The notion of electronic payments is not a new one. It dates back as far as early 1980s when David Chaum first presented the concept of using blind digital signatures for implementing untraceable electronic payments. Since then, there has always been much interest in electronic payment systems. Our research has shown that each new proposal focused on two main areas of interest: anonymity of participants of each transaction and the

security of the underlying currency representation (be it digital coins or accounts). For digital coins the primary focus has always been on solving the double-spending problem whereby the same user spends a digital coin twice. For account-based systems, anonymity has always been the more difficult problem to solve.

With the growing popularity of electronic payments a new payment model has emerged. Due to the ever-increasing demand for subscription-based services, direct debit payments have become a very popular method of payment, as is evident from annual reports of both the Reserve Bank of Australia (RBA, 2005) and Australian Payments Clearing Association (APCA, 2005). These types of payments are not confined to subscription-based services and are now available in most industries where traditionally lump-sum payments were required in the past (e.g. insurance industry).

It is no surprise that direct debit payment model found its way onto the Internet and is now being used as an alternative payment method. Fundamentally, however, direct debits are essentially a paper-based payment model. It revolves around a concept of a direct debit request (DDR) form, which is a signed, legally (if not computationally) binding contract between a customer and a merchant.

Personal experience has shown that the direct debit model currently in use online has not changed at all from its paper-based roots. As such, it is neither flexible nor secure and is open to abuse by merchants. Unlike its off-line counterpart, DDR forms are not signed and provide no binding (cryptographic or otherwise) to customers or merchants. This means that in actual fact, merchants may change the terms of DDR agreement post-fact, or in other words, disregard the agreement entirely and use customer account details in whatever way is most profitable. The Reserve Bank of Australia has acknowledged these problems as early as 2001 in its annual report (RBA, 2001), however, the changes that it recommended and introduced are policy driven rather than technical.

Clearly the use of direct debit as an electronic payment method is problematic unless major changes are introduced to restrict merchant abilities to access customer accounts. So far, no other payment scheme to our knowledge has seriously looked into this issue. In fact, most electronic payment research we have examined so far is focused on duplicating the features of physical cash and as such is not appropriate to this model.

In this paper, a fundamentally different approach to electronic payments is taken. Instead of treating each payment transaction as a point-to-point transfer of funds,

simulating physical cash changing hands, payments are viewed as a series of linked transactions. Just like in the direct debit process, customers are required to sign an agreement, which delegates the right to withdraw funds from their accounts to a nominated merchant. The key difference between this proposal and what is currently provided by direct debit is the cryptographic binding of the agreement to the customer and the merchant providing traceability and enforceability.

Another significant advantage of the new approach is its practical use of the payment agreement for enforcing the terms during every single payment transaction. Unlike with the paper-based model where the agreement is only enforceable when legally disputed, or the electronic version where it is not even traceable or binding, the new approach actively assures that each transaction is within the agreed bounds of the policy effectively preventing fraud.

The periodical payment model presented in this paper is driven by the X.509 restricted proxy certificate standard. Restricted proxy certificates are used to delegate permissions to transfer funds from customer accounts to merchants. This certificate becomes the binding artefact that links the customer and the merchant together and that could be used in case of disputes.

Finally, a payment policy language is presented that provides the main instrument for instantaneously enforcing each transaction as discussed previously. This language was designed to describe most common scenarios that are applicable to periodical payments but of course it is flexible enough to accommodate traditional, not periodical payment types as well.

The next section presents a brief overview of the most relevant electronic payment schemes. It also discusses the grid security architecture that successfully used proxy certificates for delegation. Then, follows a detailed introduction of the periodical payment model broken down into two sections: 1) the initial delegation of the payment credential to the merchant and 2) the use of that credential to initiate a single payment transaction. The semantics of the payment policy language will be discussed next. Finally, a brief discussion of anticipated future work is presented.

2 Related Work

There are numerous electronic payment systems that have been proposed and developed over the years. However, none approach the problem from the standpoint of periodical payments. As such there is very little work that has directly influenced the direction of this paper.

Most of the work that deals with notational, account-based electronic payments has focused on three major areas of research: authentication of participants, security of transactions and in some cases anonymity. No explicit attempts to solve the problem of delegation in electronic payment environments have ever been seriously considered.

There is some research that stands out as relevant in their approach to payments in general and could be considered

as the first building blocks for the implementation of periodical payments. For example, (Bellare et al., 1995) and (Bellare et al., 2000) presented an interesting protocol entitled i-Key-Protocol (iKP), which discussed the possibility of integrating a new secure approach to payments into an existing payment clearing infrastructure. Just like the proposal discussed in this paper, iKP uses the payment gateway concept as the mechanism for interacting with merchants. Its purpose is to convert payment artefacts into format that will be recognised by the legacy systems.

Similarly, iKP also relies heavily on certificates as the authentication and authorisation mechanism. However, the authors do realise that the level of PKI acceptance and market penetration is not sufficient to allow effective adoption of the mechanism. Hence this protocol introduces a staged integration approach whereby the first stage requires no PKI at all while the later stages gradually introduce certificates.

Another interesting notational scheme is the anonymous Internet mercantile protocol presented by (Low et al., 1994). This protocol does not address the unique requirements for periodical payments directly, however, it is one of the very few schemes that deliver a form of multi-transactional support. It works by allowing customers to set up session (i.e. temporary) accounts with merchants, which can be debited by the merchants without explicit involvement of the customers. Whenever the funds in such accounts run out, the customer is responsible for injecting more money if the relationship with the merchant is to continue.

The main motivation for designing the mercantile protocol was not to enable periodical payments. Instead it arose from the desire to streamline the payment process so that it would be quicker and more convenient for the customer and merchant.

There are significant conceptual differences, however, between the protocol presented by (Low et al., 1994), and the one discussed in this paper. One major limitation of the session account concept is the fact that all of the funds have to be committed before the transaction takes place. This, in a fact, contradicts our current definition of periodical payments, which allows customers to split a single large payment into multiple smaller ones, distributed across multiple payment periods. This is important since in some cases the motivation for choosing periodical payment option is because of the lack of funds to complete the entire transaction in one hit.

Since the funds are committed at the beginning of the transaction, this protocol therefore fails to deliver on another important requirement for periodical payments. The customer is required to relinquish control of the funds to the merchant and loses all control of how those funds are managed. This means that given a malicious merchant, the funds can be taken without permission. No effort to restrict access to funds is made with the only advisable safeguard being to transfer only small amounts of money into such accounts.

Due to the lack of appropriate solutions to the periodical payment problem, the work on grid security was closely

studied so that some of the underlying concepts could be adopted for use in electronic payments. For example, (Welch et al., 2004) presented a discussion on the use of X.509 proxy certificates for enabling delegation in grid environments. This work followed the general discussion on grid security services presented by (Welch et al., 2003).

Significant amount of research and development towards improving and standardising the proxy certificate specification (Tuecke et al., 2004) was accomplished through the development of the grid security services. Its major contribution, the Open Grid Services Architecture (OGSA), and in particular, its Grid Security Infrastructure (GSI), has delivered a set of standard tools that use proxy certificates for authentication within a complex, heterogeneous environment.

The grid architecture presents a very compelling reason for using proxy certificates since it requires its users to authenticate themselves to various services distributed across a wide area network. Furthermore, its sole purpose is to allow users to execute computationally expensive operations, which can potentially take hours, or days and that require minimal supervision from those users. When presented by (Koufil and Basney, 2005), these tasks were commonly referred to as batch jobs because they are not executed when submitted but are scheduled and executed when appropriate hosts are found and enough resource have been allocated to execute them.

It is clear how the use of proxy certificates within grid environments is applicable in periodical payments. Periodical payment requirements on delegation and authorisation can be seen as a subset of the overall problem that the grid security infrastructure is trying to solve. The general principal behind the use of restricted proxy certificates is to delegate both the issuer's identity and some subset of issuer's privileges to the bearer. For periodical payments, the resources for which access is granted is always the customer's bank account, while the privileges being delegated always refer to the amount of funds that can be transferred and who can do so.

Some common issues that are relevant in grids are not relevant in periodical payments, which simplify the paradigm. For example, dynamic creation of new entities within the network, which is an important concern in grids, is not an issue since it is always known before each transaction, which parties are involved and once the payment contract is negotiated no amendments are allowed.

In the next section, a complete model of the periodical payment process is presented. This scheme uses the restricted proxy certificates developed as part of the grid security project but instead of authentication their role is to convey customer account access authorisation information.

3 Periodical Payment Process Using Restricted Proxy Certificates

The complete periodical payment process is somewhat involved. Unlike the traditional model, which assumes

immediate transfer of funds, periodical payments do not. Instead, this process works in a schedule-type framework, whereby the initial (and only) customer-merchant interaction only establishes the terms of each scheduled transaction. Before discussing the details of the periodical payment protocol it is important to understand this fundamental idea.

The periodical payment policy is the core of the model. This policy is essentially an agreement, a contract between a customer and a merchant. The merchant promises to provide a customer with a service while the customer agrees to pay for that service on regular basis. This policy is essential to this process because it places restrictions on the delegated credentials that the customer must give the merchant so that all subsequent transactions can be performed without further customer intervention.

The important qualities for the policy language that are of particular interest are its simplicity and flexibility. It is envisaged that the contents of the payment policy would need to be presented to customers in a human-readable form so that manual validation could be performed. As such, language simplicity is of the highest importance to ensure that customers can easily visually inspect each periodical payment contract and understand it.

While simplicity is of particular importance, language flexibility is also crucial. The policy language must contain sufficient expressiveness to describe most common periodical payment scenarios. For example, when delegating credentials, it is most likely that customers would be interested in the following restrictions:

1. The merchant will only charge customer account for services provided (or to be provided) to the customer.
2. The merchant will only charge the agreed amount or within a specified range (e.g. anything under \$200, etc.)
3. The merchant will only charge customer account once per agreed payment period.
4. The merchant will cease to charge customer account upon completion of the contract or as soon as the service for which it was established is no longer provided.
5. The merchant will cease to charge customer account on request.

It is clear how the above assertions could be used by a payment gateway to validate each transaction initiated by the merchant using customer-delegated credentials. Later in this section, it will be demonstrated how each of the above assertions can be easily expressed using our experimental policy assertion language. For now, let's examine the two stages of the periodical payment process: 1) the delegation of the payment credential (including signing of the policy document), and 2) initiation of a payment transaction (i.e. transfer of funds from customer to merchant account).

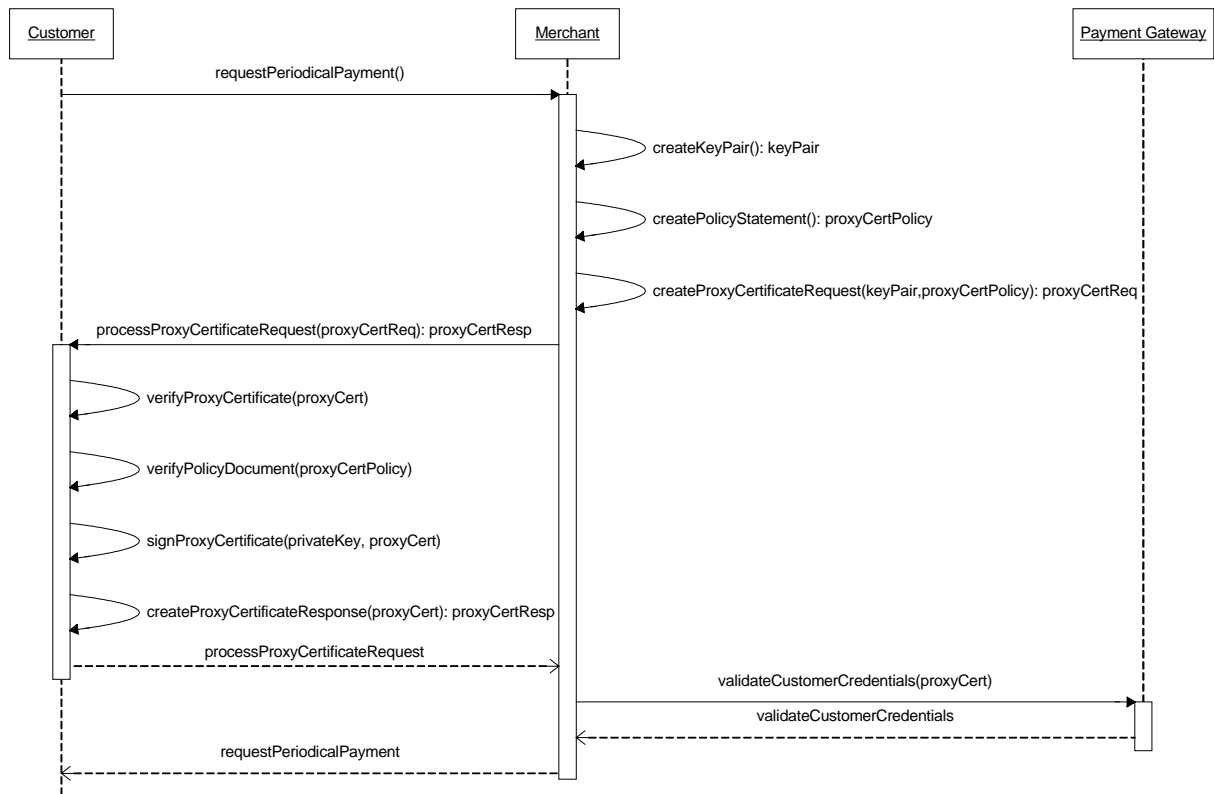


Figure 1: Payment Certificate Delegation Process

3.1 Delegation of Periodical Payment (Proxy) Certificate

The delegation of a periodical payment certificate is the first step in the payment process and is the only one that involves the customer. It is based on the X.509 proxy certificate delegation process defined in (Tuecke et al., 2004). The only addition to the standard protocol that is necessary is the creation and acceptance of the policy document that forms the foundation of the payment certificate. The delegation steps are depicted in Figure 1 and can be described as follows:

1. Once the customer requests a periodical payment option, the merchant generates a new public-private key pair.
2. The merchant then creates a policy statement.
3. Using the key pair created in step 1, the merchant creates a restricted proxy certificate request. This request must conform to the specification described in (Tuecke et al., 2004). It will also contain the policy created in step 2.
4. The request is sent to the customer for consideration (i.e. the customer must process the request to examine the policy and to sign the certificate).
5. The customer must verify that the proxy certificate request is valid, i.e. it is not an end-entity certificate and all of its mandatory fields have been correctly set. Normally, the policy document is an optional field, however, for our purposes it is mandatory.
6. The customer must review and verify the terms of the policy document received from the merchant. It can either try to renegotiate the terms of the policy, reject it or proceed with the next step.
7. The customer must sign the proxy certificate with its own private key, create a response message and send it back to the merchant.
8. Finally, the customer can create the proxy certificate response message and forward it to the merchant.
9. Once the merchant receives the response, it needs to validate the authenticity of its signature. This task is delegated to the payment gateway that is capable of validating all customer credentials. If a positive response is received, the merchant can use the proxy certificate subject to its policy.

The final step (step 9) of this process concludes with the merchant obtaining possession of the restricted proxy certificate which, when presented to a payment gateway, will enable it to transfer funds from customer account to its own. The policy contained within this certificate is the artefact that is used by the payment gateway to make its decision whether to allow the transfer to proceed or to reject it.

There is a significant complication with the above process. Upon detailed analysis it was observed that a certificate with a restriction policy is not sufficient for the payment gateway to determine whether a single transaction is within the bounds of the policy agreement. That is, since a policy can dictate a long period between each transaction (e.g. a month or a quarter), a policy alone does not convey the necessary information to

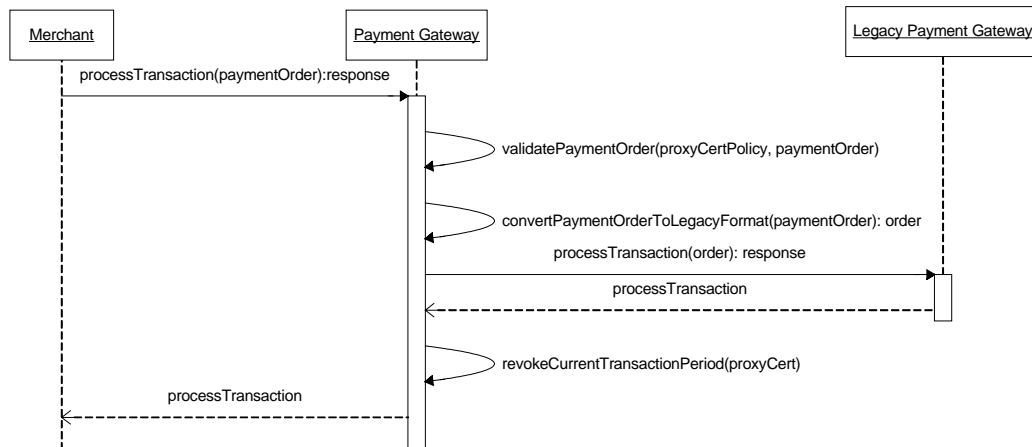


Figure 2: Single Payment Transaction Process

uniquely identify a transaction. This is a classic double-spending problem. In this case, the payment gateway cannot use a certificate policy alone to determine whether a particular transaction was already performed within a specific payment period.

3.1.1 Solving the Double-Spending Problem

Clearly the double-spending problem as discussed so far is precisely the reason why a new periodical payment approach is being presented. It is one of the most important issues that cannot be sufficiently addressed by the current direct debit model.

Using proxy certificates for payment authorisation, however, provides the periodical payment model with various convenient ways of solving this problem. The simplest solution to the problem is for the payment gateway to record each transaction performed by the merchant. This log of transactions in combination with the certificate policy is sufficient to determine whether an individual transaction is valid. The payment gateway must simply ensure that all assertions within the policy are met and that the merchant has not performed a transaction using this certificate in this payment period by consulting the transaction logs.

This solution demands a great deal of the payment gateway. It is inefficient to store all transactions that have been performed and after closer analysis it was observed that it is not necessary either. The payment policy, which will be presented in depth later in this paper, declares one important element called *period*, which can help a great deal in reducing the amount of data that a payment gateway must store.

The *period* element declares the interval between each transaction (e.g. week, month, bi-monthly, etc.). This is precisely the information that the payment gateway needs when logging transactions.

The design that was adopted for the logging mechanism is based on the concept of revocation similar to certificate revocation lists described in (Housley et al., 2002). The full credential cannot be revoked since no more transactions would be possible and the periodical

payment model will be broken. Instead, the payment gateway should only be interested in the last transaction that was performed by the merchant using a payment credential. Since the credential cannot be revoked, the payment gateway can use the *period* value declared in the policy to revoke the use of that credential for that period. When used a second time within the same period, the payment gateway can check its revocation lists and identify double-spending attempt.

The format of the revocation lists for payment certificate periods is simple. It needs to contain two values. The first value must be the X.509 certificate unique identifier and the second the period expiry date. The period expiry date is the last date when the policy can be used to access customer funds within a particular period. The only complexity is the determination of the expiry date based on the policy period attribute. However, this is not a major issue since the payment gateway can reuse its policy validation process, which must perform a similar function, to extract this date.

3.2 Payment using Periodical Payment (Proxy) Certificate

The actual process of initiating a payment transaction is simple. The payment certificate, which is just a normal X.509 proxy certificate can be passed to the payment gateway via Transfer Layer Protocol (TLS) establishing a secure connection to the gateway server. No modification to this protocol is needed. It will handle all of the necessary proxy certificate authentication-authorisation using the standard path validation process described in (Housley et al., 2002). Post authentication process is depicted in Figure 2 and can be described as follows:

1. Once an encrypted channel is initialised, the merchant can send the payment order for processing of the transaction. This can be a simple XML object that contains payment details such as from and to accounts and amount.
2. Having received this information, the payment gateway must validate it (i.e. validate payment order)

against the policy document it received via a payment certificate. It must verify that:

- The certificate has not expired.
- This transaction is within an acceptable period as declared in the policy.
- The current payment period for this certificate has not been revoked, that is the merchant is performing this transaction for the first time within this payment period.
- The payment order amount is within the acceptable range as declared in the policy.

Any errors in validation will force the payment gateway to reject the payment order.

3. Upon successful validation of the payment order against the policy, the payment gateway can proceed with its processing. At this time, it is assumed that an existing payment clearing infrastructure is most likely to be used for actual funds transfer. As such, the role of the payment gateway in this scenario is to convert the payment order into a format that the legacy clearing infrastructure can understand.
4. Converted payment order is then submitted to the legacy system. Clearly in this case any problems that occur during payment processing once submitted are out of scope.
5. The final step that the payment gateway must perform is to revoke the current payment transaction period for the certificate. Clearly this step cannot fail, otherwise this will give a merchant an opportunity to submit a duplicate payment order and charge its customers twice. For this reason, great effort is required to assure that payment submission and the revocation process are atomic operations.

Upon completion of this process the payment gateway may respond to the merchant by issuing a response (i.e. receipt) message that it can use as a reference.

4 Periodical Payment (Proxy) Certificate Policy Language

Periodical payment policy language is expressed as an XML document and it is surprising simple. It requires only three custom data types with just four different XML elements. Each element represents an assertion that the payment gateway must evaluate to true before processing a transaction.

Assertions By Type	Element
Date	<not-before-date> <not-after-date>
Periodical Transaction	<transaction>
Payment	<payment>

Table 1: Policy Elements

The first and most straightforward assertion type is the date type. Naturally, its purpose is to provide a mechanism for specifying concrete dates that can be used

to declare contract boundaries where appropriate. Within the policy XML this type is represented by two elements: <not-before-date> and <not-after-date>. Each element contains a single attribute, *value*, whose contents must conform to the XML date format defined by the (XMLSchema, 2005) specification.

Attribute	Required?	Value
Value	Yes	xsd:date

Table 2: Date Attributes

For example:

<not-before-date value="2007-01-01"/> and

<not-after-date value="2007-12-31"/>

The date assertions have an important role within the policy. Not only can they be used as global assertions that specify the contract boundaries, they can also be used as constraints placed on other assertions within the same policy. Constraints will be examined in more detail later in this section.

In addition, to the date assertion type, the policy declares a payment type. This type is expressed by a single XML element <payment>. Its purpose is to define the payment options that are available to the merchant. That is, using this assertion the payment gateway can determine whether the amount of funds that the merchant wishes to transfer from the customers' account is within the agreed range.

The payment assertion is capable of representing various amounts and currencies. It can be used to declare a single, non-changing amount or it can specify a range of acceptable amounts. To achieve this, it uses three attributes: *currency*, *amount* and *type*.

Attribute	Required?	Value
Amount	No	xsd:decimal
Currency	Yes	AUD USD ...
Type	Yes	Fixed Limit No-limit

Table 3: Payment Attributes

The purpose of the currency attribute is self-explanatory. Since it is possible for transactions to be international, a mechanism is required to specify which currency the merchant is dealing with.

The amount attribute is closely related to the value of the type attribute. Depending on whether the type is *fixed* or *limit*, the meaning of the amount attribute changes. For a fixed type, the amount value is static. That is, the merchant can only withdraw the exact amount declared in the policy. Anything else must cause a validation error. For example:

<payment currency="aud" amount="20" type="fixed"/>

For limit type, on the other hand, the amount value indicates the upper boundary of the acceptable range of values. In this case, anything up to and including the value is considered acceptable. For example:

```
<payment currency="aud" amount="100" type="limit"/>
```

Finally, the last and undoubtedly the most complicated assertion type is the periodical transaction type. Within the policy XML this type is represented by a single element `<transaction>`. Its purpose is to declare the interval (i.e. period) between each allowed transaction. It must do so within a single expression, which makes it slightly more complicated than the previous two assertions.

It is clear why the transaction assertion is complex. Within a single rule it must be able to represent the complex behaviour that is inherent in periodical payments. It must be flexible enough to describe all transactions within the scope of the contract. As such, this assertion requires four attributes: *period*, *day-of-week*, *week-of-month*, and *day-of-month*.

Attribute	Required?	Value
Period	Yes	Daily, Weekly, Fortnightly, Monthly, Quarterly, Half-yearly, Yearly
Day-of-week	No	[Mon, Tue, ... Sun]
Day-of-month	No	[1...31]
Week-of-month	No	[1...5]

Table 4: Transaction Attributes

The period attribute has already been briefly introduced earlier in this paper. Its purpose is to indicate the agreed frequency of transactions, for example weekly, monthly, quarterly, or yearly. By itself, however, it is not sufficient. More information is required by the payment gateway to determine when precisely a transaction can be processed.

Day-of-week, week-of-month and day-of-month are the attributes that can be used to provide the payment gateway with that extra information that it needs. Their use is determined by the value of the period attribute. For example, for a weekly period, only the day-of-week attribute is required. Its value must be a valid weekday three-letter acronym. For example:

```
<transaction period="weekly" day-of-week="mon"/>
```

For a monthly period, on the other hand, day-of-month attribute can be used to specify on what day of every month a transaction can be processed. For example:

```
<transaction period="monthly" day-of-month="1"/>
```

As you can see there are various possible combinations, each one requiring the use of different attributes.

4.1 Coping with Odd Transactions

The fundamental principle underlying the payment and transaction assertions is that there will only be a need for one assertion each to describe all of the necessary

conditions for a contract. This assumption is true in most cases; however, there could be instances when a policy needs to declare an odd transaction to cope with a special case. For example, for a fixed term contract, there could be a need to declare an odd transaction to handle the repayment of the remaining amount upon termination of the contract (i.e. as part of the last transaction).

To handle odd transactions additional payment or transaction assertions need to be declared within a single policy. Having two or more assertions of the same type within one policy, however, is problematic because it introduces ambiguity as to which one the payment gateway should validate against for a particular payment period. To remove this ambiguity, the concept of constraints was introduced.

A constraint is not a stand-alone assertion. Instead, it is an optional sub-element of the payment or transaction assertions. Within the policy language it is expressed as an XML element `<constraints>`. Within each constraints element at least one date assertion must be declared, either `<not-before-date>` or `<not-after-date>`, or both. When validating a payment order against a policy the payment gateway can distinguish between assertions of the same type by checking whose constraints make it applicable within the current payment period.

The following example depicts two payment assertion declarations whose constraints define which payment period they apply to:

```
<payment currency="aud" amount="80" type="fixed">
  <constraints>
    <not-after-date value="2007-11-30"/>
  </constraints>
</payment>
<payment currency="aud" amount="50" type="fixed">
  <constraints>
    <not-before-date value="2007-12-01"/>
  </constraints>
</payment>
```

By introducing the concept of constraints additional policy validation logic is needed. This checking is important because even with constraints it is still possible to declare two assertions of the same type that overlap within a payment period. Policy validation must identify this scenario and reject any contract that does not uniquely specify a single assertion per payment period.

5 Periodical Payment Model Implementation

In this section we shall briefly describe periodical payment components that have either already been implemented or are due to be implemented in the near future.

5.1 Periodical Payment Policy Parser and Validation

To date, the first draft of the policy XML schema has been completed. Its corresponding parser and validation logic have been implemented using Java and Apache XML-Beans toolkit. Using XML-Beans allows us to bind

XML types to Java objects. XML-Beans is used to compile the policy XML schema that generates both the XML parser and the corresponding Java data types.

When parsing policy XML contracts the parser performs semantic/syntactic validation of the XML. However, this validation is insufficient. Additional validation logic was implemented to check policies for ambiguity. A custom validator, written in Java, is used for checking that all elements that have constraints are not in conflict with each other. In addition, element attributes that are inter-dependant are checked to ensure that their values are meaningful in the context in which they are used.

5.2 Customer and Merchant Libraries

Periodical payment process is composed of two distinct communication components. The first component dictates the communication between the customer and the merchant while the second one controls merchant-payment gateway interaction. Considering that our primary focus to date was the XML policy schema specification and its validation, at the time of writing the communication components have not been implemented.

For the customer-merchant interaction client-side libraries are needed that can securely communicate with the merchant payment systems. Unfortunately, currently there are no standard mechanisms for implementing such client libraries. We are considering implementation of a browser plug-in that will detect periodical payment transactions and will facilitate customer interaction with merchant payment systems.

Implementation of the merchant to payment gateway interaction component will be completed in the near future using web services. Web services technology offers two important advantages. Firstly, this technology integrates well with the secure socket layer (SSL) protocol. We heavily rely on it for establishing a secure channel between a merchant and a payment gateway. Also, it is the mechanism that allows the merchant to send the policy statement to the payment gateway via a proxy certificate.

In addition, web services are a proven technology. They have been used in numerous commercial applications where decoupling of components and interface flexibility is of particular importance. Unlike existing payment gateway interfaces, using web services allows us to generalise this interface, which means that merchants will no longer require proprietary libraries supplied by the payment gateway providers to process transactions.

Existence of readily available web services implementations gives us confidence that this technology can be quickly adopted for periodical payments. Its performance, however, is a crucial factor that will impact its acceptance as an electronic payment solution. Therefore, it is our intention to develop a prototype implementation of the payment gateway web services interface, and to conduct initial performance testing of it to validate that it can process a suitably large number of concurrent transactions.

6 Future Work

As was mentioned in the previous section, the next task on our agenda is the development of the payment gateway web services interface and corresponding merchant libraries. Using this implementation we shall analyse the performance of the payment model. In addition, the necessary customer side libraries will also be developed to allow for complete, end-to-end test coverage of the payment process.

The periodical payment model is a complex scheme that requires infrastructure-wide changes to be effective. Presently, it is unreasonable to expect all consumers to be able to properly protect their private keys. The adoption of smart cards is slow and most current computers are not equipped to read them. As such, it is unlikely that any technology that makes demands on client side PKI will work. Therefore, as an alternative, it will be examined how conventional username/password technics can be adapted to the periodical payment model presented here.

Another potential area that needs closer scrutiny is converting the periodical payment policy into human readable form. There are two important reasons for this. Firstly, as it was stated previously, we consider the customer's ability to visually inspect the policy contract an important part of the policy validation process. Unless this information is presented to a customer in an understandable way, the customer's signature will have no meaning.

Another improvement that is closely related to the one above, is exploring mechanisms for customers to negotiate the terms of the policy contracts. Once each policy is presented to the user in a readable form, a mechanism is needed that allows the customer to make changes to the policy and submit them to the merchant for consideration. Merchants need ability to declaratively specify the boundaries of acceptable amendments so that change requests can be automatically processed and accepted, rejected or alternative counter-amendment proposed. A communication protocol needs to be established that allows both parties to participate in this negotiation in a secure manner.

Also, an interesting area that needs further improvements is the payment gateway transaction validation process. The current model requires the payment gateway to keep a transaction revocation list to prevent double spending, which is an expensive task. Ideally, the payment gateway should be stateless and require no local storage of transaction data. This means that further investigation needs to be performed in determining whether more information can be encoded in the payment credentials that will reduce the need for local data to be maintained by the payment gateway.

In addition, it should not be overlooked that regardless of its intended purpose, the periodical payment model is just another payment scheme. As such, some investigation is warranted into its application as a conventional, non-periodical payment scheme. Of course depending on the overall complexity of using this particular approach as a normal payment mechanism it may prove impractical.

Finally, even though periodical payments do not fit well the traditional electronic payments approach it is still quite interesting to examine potential addition of anonymity to the scheme. In most electronic payments research a great deal of importance is placed on anonymity and as such should not be overlooked for periodical payments although once again their practical use might be of limited interest and acceptance.

7 Conclusion

Periodical payments are radically different to traditional e-commerce payments. Payment transactions require no customer involvement during each transaction and empower the merchant to initiate payment transactions at whichever time they see convenient. Transfer of control of customer personal banking details to merchants creates an overwhelming need for a better security mechanism than what is currently available.

Periodical payment model represents an important change in direction for electronic commerce payments. They fill the security and useability gap that the direct debit model used now is incapable of doing due to its paper driven design. Neither can the existing solutions offer value due to their fundamental focus that place anonymity and non-relinquishing of control as priority.

The periodical payment model presented in this paper builds on a strong cryptographic foundation. It uses the existing and proven X.509 standard for securing payment channels and for providing customer, merchant and payment gateway authentication. The use of the restricted proxy certificate concept allows us to implement transfer of control mechanism that enables the customer to securely transfer account access rights to the merchant.

The use of restricted proxy certificates serves another purpose. It can be used to carry the periodical payment policy. This policy replaces the traditional direct debit request form as the customer-merchant payment agreement that the customer must sign. Our ability to easily verify the legitimacy of the payment certificate and to determine the applicability of policy assertions to the current transaction is what gives this model its great flexibility and allows safe transfer of control through delegation.

8 References

- APCA (2005): Annual Report. Sydney, Australian Payments Clearing Association.
- Bellare, M., Garay, J., Hauser, R., Herzberg, A., Krawczyk, H., Steiner, M., Tsudik, G. & Waidner, M. (1995): iKP - A Family of Secure Electronic Payment Protocols. *Proceedings of the First USENIX Workshop on Electronic Commerce*. New York, USA, USENIX.
- Bellare, M., Garay, J. A., Hauser, R., Herzberg, A., Krawczyk, H., Steiner, M., Tsudik, G., Van Herreweghen, E. & Waidner, M. (2000): Design, implementation, and deployment of the iKP secure electronic payment system. *IEEE Journal on Selected Areas in Communications*, 18, 611-627.
- Housley, R., Polk, W., Ford W. & Solo D. (2002): Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. *RFC3280*. Internet Engineering Task Force (IETF).
- Koufil, D. & Basney, J. (2005): A credential renewal service for long-running jobs. *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*. Seattle, Washington, USA, IEEE Computer Society Press.
- Low, S. H., Kristol, D. M. & Maxemchuk, N. F. (1994): Anonymous Internet Mercantile Protocol. *Technical Report*. Murray Hill, New Jersey, AT&T Bell Laboratories.
- RBA (2001): Payments System Board. *Annual Report*. Sydney, Reserve Bank of Australia.
- RBA (2005): Bill payments and Automated Teller Machines. *Annual Report*. Sydney, Reserve Bank of Australia.
- Tuecke S., Welch, V., Engert, D., Pearlman, L. & Thompson, M. (2004): Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile. *RFC3820*. Internet Engineering Task Force (IETF).
- Welch, V., Foster, I., Kesselman, C., Mulmo, O., Pearlman, L., Tuecke, S., Gawor, J., Meder, S. & Siebenlist, F. (2004): X.509 Proxy Certificates for Dynamic Delegation. *Proceedings of the 3rd Annual PKI R&D Workshop*. Gaithersburg MD, USA, NIST Technical Publications.
- Welch, V., Siebenlist, F., Foster, I., Bresnahan, J., Czajkowski, K., Gawor, J., Kesselman, C., Meder, S., Pearlman, L. & Tuecke, S. (2003): Security for Grid services. *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*. Seattle, Washington, USA, IEEE Computer Society Press.
- XMLSchema (2005): XML Schema Specification, World Wide Web Consortium (W3C), <http://www.w3.org/2001/XMLSchema>. Accessed 3 Aug 2006.

Mutually visible agents in a discrete environment

Joel Fenwick

V. Estivill-Castro

Institute for Integrated and Intelligent Systems
Griffith University, Australia

Email: j.fenwick@student.griffith.edu.au v.estivill-castro@griffith.edu.au

Abstract

As computer controlled entities are set to move and explore more complex environments they need to be able to perform navigation tasks, like finding minimal cost routes. Much work has been done on this problem with a single entity in a continuous environment. However these entities may be in teams and their actions may be constrained by the need to consider the actions of other entities. We consider the case of two entities wishing to reach their destinations while travelling the minimum distance and remaining in sight of each other. A version of this problem in continuous space has been addressed previously; however, the problem of minimum length paths does not only exist in continuous space. Robots may have restricted orientations in movement. Also, domains such as circuit design and computer games require discrete movements and restricted orientation. The restricted orientation and the fact that (even in genus zero environments) minimal length paths may not be unique present some challenges. This paper investigates the problem of two entities in a discrete setting, moving to preserve visibility and provides algorithms for computing schedules that minimise the total distance travelled.

1 Introduction

There has been significant interest in problems relating to visibility and/or shortest paths in polygonal environments due to the large number of applications in areas like computer graphics, robot navigation, geographical information systems and circuit layout. From the computational geometry perspective, these are presented as path planning problems (LaValle 2006) or guarding a region problems (O'Rourke 1987), and together with graph algorithms, the literature has seen many advances (Sack & Urrutia 2000). However, most of these advances concentrate on finding one shortest path within continuous (vectorial) representations, perhaps with some constraints, and possibly with restricted discrete orientations. Little work has been focused on cases with two or more entities where movement is to be determined under some visibility constraint. Less work falls into the intersection of finding several paths when the environment is discrete or enforces some sort of restricted set of orientations for movement (Fink & Wood 2003, Flochini, Prencipe, Santoro & Widmayer 2005, Schuierer & Wood 1999, Widmayer, Wu, Schlag & Wong 1986).

Copyright ©2007, Australian Computer Society, Inc. This paper appeared at the Thirtieth Australasian Computer Science Conference (ACSC2007), Ballarat, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 62. Gillian Dobbie, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

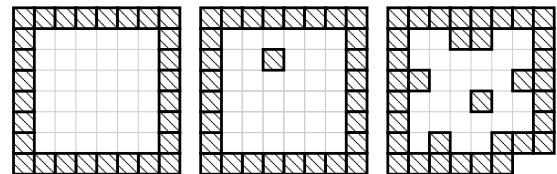


Figure 1: Inputs with the same sized bounding box.

We consider the case of two entities in a polygonal environment wishing to reach their destinations while travelling the minimum distance and remaining in sight of each other. A version of this problem in continuous space has been addressed (Fenwick & Estivill-Castro 2005); however, the problem of minimum length paths does not only exist in continuous space. Robots may have restricted orientations in movement. Also, domains such as circuit design and computer games require discrete movements and restricted orientation. Other work (Ickling & Klein 1992) in the continuous case has discussed two guards maintain mutual visibility while walking on respective sides of a polygon's boundary.

We note that, in order to formally discuss the complexity of the discrete problems, and therefore of algorithms for solving them, some consideration must be given to the specification of input. Perhaps because less research has been performed on discrete environments, there is no widely accepted format for the input as for the continuous case (where a simple polygon is provided as a sequence of n points in the plane and commonly accepted as having input size n (de Berg, van Kreveld, Overmars & Schwarzkopf 2000, Preparata & M.I. 1985, O'Rourke 1994)). In particular, the inputs we consider here could be expressed in discrete form (either as a list of border cells or a description of the state of all cells) or in continuous form as sequences of border points and a scale factor. The different possible forms to describe the input lead to different ideas about the size of the problem instance. Figure 1 illustrates the difference. When measured in terms of vectors, the input sizes are 4 points, 8 points and 26 points. If the centres of the cells are used instead, we have 4 points, 5 points, approximately 21 (depending on the precise encoding scheme). If the input is supplied as the state (interior cell vs exterior cell) of each cell in a rectangular region, then all the instances have the same size input size. They all have similar size if a list of those forbidden (exterior) cells that constitute the boundary is supplied. For the most part of this paper we will accept the input as a grid with cells labelled as belonging to the interior or (exclusive) to the exterior. This is a common representation in computer games (Davis 2000). We accept that the input could be translated to other formats in polynomial time,

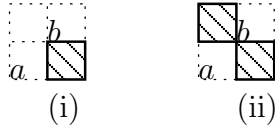


Figure 2: Navigable corners: (i) Cell b is reachable from a in one 8-connected move or two 4-connected moves. (ii) Cell b is not reachable from a in two 4-connected moves and is deemed not to be reachable in a single 8-connected move either. However, a and b may still be connected if there is a longer 4-connected path around the obstacles from a to b .

but not necessarily linear time. While our convention allows us to test a cell as belonging or not to the polygon in constant time, we will attempt to express our algorithms in terms of such a test, and to describe the complexity as a function of the complexity of this test. This would allow for other input formats.

This paper considers the discrete version of a problem where two agents must travel on minimal length paths while maintaining visibility. We provide the precise definitions and statement of the problem in Section 2. We take the opportunity to illustrate there that identifying if shortest paths form a solution is significantly different from the continuous case. Section 3 provides an algorithm for discrete polygons with or without holes. This algorithm is based on transforming the problem to finding connectivity in a graph of states. Therefore, the complexity is construction of the graph plus breadth first search (or Dijkstra’s algorithm) on such a graph. In Section 4 we show an avenue to reduce this cost by using a smaller state graph based on the Euclidean shortest path. However, this restricts the algorithm to polygons without holes. We also characterise a subset of problem instances that always have a solution. This characterisation allows us to build decompositions of the general problem such that, if a simple reduction applies, it can be solved optimally.

2 Problem Description

Definition 1 A cell is a square region of \mathbb{R}^2 with unit sides. A point belongs to the cell if it is inside or on the border of the region. A cell is identified with its centre point.

Definition 2 A discrete polygon consists of a 4-connected set of cells in a rectangular grid surrounded by border cells. Unlike a “vectorially described” polygon, these border cells are not considered to be part of the polygon.

Note that this 4-connected constraint is imposed on 8-connected problems as well. Figure 2 illustrates this. Cells are either interior cells or boundary cells.

Definition 3 Two interior cells are visible if the line segment joining the centres of the cells does not intersect the interior of any boundary cells. Such a segment is called a sightline.

Note that sightlines are permitted to touch a corner or the side of boundary cells. In the pattern recognition literature, visibility has been defined for discrete environments (a grid of pixels) in terms of an 8-connected path that is the rasterization a straight line segment joining the cells (Coeurjolly, Miguet & Tougne 2004, and references). We will not use this notion of visibility for this paper; however, we point out that there are efficient algorithms to test if two cells are mutually visible when the environment is provided as cells labelled as interior vs exterior (Coeurjolly et al. 2004).

Definition 4 A discrete path in a discrete polygon is a (either 4 or 8)-connected sequence of interior cells beginning with the “start cell” and ending with the “target cell”. A path is minimal if it has minimal length, where length is the smallest number of cells visited from the start to the target cell less one (or the smallest number of jumps).

Note that we can compute the minimal continuous path in a discrete polygon, since such a polygon can be treated as a continuous simple polygon. The minimal continuous path in a polygon is a polygonal line (Guibas & Hershberger 1987). Also, in discrete environments, discrete geodesic shortest paths have been defined (Coeurjolly et al. 2004, and references) so that these paths are 8-connected curves that are segmented into Discrete Straight Segments (those 8-connected paths that are rasterization of a straight line between two cells). For the purposes of this paper, we do not use such notion of discrete path, as here we intend to reflect the motion restriction, rather than to emulate a continuous domain with a grid.

We are now in a position to formulate the problem.

Definition 5 A discrete mutually visible path problem (DMVPP) consists of a discrete polygon P and four distinguished interior cells $\vec{s}_1, \vec{s}_2, \vec{t}_1, \vec{t}_2$. The line \vec{s}_1, \vec{s}_2 and the line \vec{t}_1, \vec{t}_2 are sightlines. A solution to a DMVPP consists of discrete paths μ_1 (from \vec{s}_1 to \vec{t}_1) and μ_2 (from \vec{s}_2 to \vec{t}_2) and a schedule (movements for each agent at each step) such that agents a_1, a_2 move from \vec{s}_i to \vec{t}_i without losing visibility.

Movement between cells is considered instantaneous with uniform cost. Visibility is only evaluated with respect to cell centres. In cases where the type of movement matters we will use DMVPP⁴ or DMVPP⁸ to disambiguate the discrete paths allowed.

We will use the following additional notation. Discrete paths will be denoted M_i or μ_i while continuous paths will be denoted π_i or Π_i (the second only being used for paths of minimal Euclidean length).

2.1 Distinctive Cases

One distinctive aspect from the continuous Euclidean setting is that the metrics in use here admit multiple paths of minimal length. In the continuous Euclidean case, it is always possible to find a schedule and paths of as short as within any $\epsilon > 0$ of the sum of the minimal path lengths (Fenwick & Estivill-Castro 2005). Moreover, if the start sightline and the target sightline do not cross and a solution exists, the minimal Euclidean paths always admit a schedule (Fenwick & Estivill-Castro 2005). However, we show here that there does not always exist a schedule for any pair of minimal paths in a DMVPP. Figure 3 illustrates this. If the points \vec{t}_1, \vec{t}_2 coincide with \vec{s}_1 , the minimum path length is 6. The figure has highlighted a minimal path which preserves visibility. However there are paths of length 6 which pass through cell y which is not visible to \vec{s}_1 .

Perhaps it is more reasonable to consider metrics that charge slightly more for diagonal steps. This discourages unnecessary zigzagging in minimal paths. While the set of minimal paths becomes more restricted, such metrics would not however, deal with the issue of minimal length paths which must contain diagonals but in which the position of those diagonals is not constrained (Lovell & Fenwick 2004). Again, Figure 3 shows that minimal paths through y fail to maintain visibility while equivalent minimal paths with later diagonal steps do succeed and all paths would have the same cost even if diagonal steps are more costly than vertical steps.

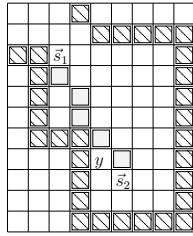


Figure 3: In the discrete, case minimal paths are not always part of a solution. The dark shaded cells are border cells while the light shaded cells show a valid shortest path between \vec{s}_2 and $\vec{t}_2 = \vec{t}_1 = \vec{s}_1$.

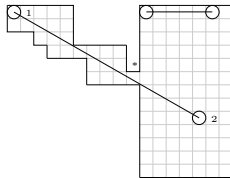


Figure 4: This DMVPP⁸ has a solution, but no solution which uses minimal length paths (a_2 moves to the gap marked *, allowing a_1 to come out).

Figure 4 illustrates more challenges of the discrete case. This DMVPP admits a schedule and paths, but no path of minimum length has a schedule for a solution. Moreover, Figure 5 illustrates a DMVPP which has no solution at all. It is not surprising that the DMVPP is much harder than the continuous one, because finding Euclidean shortest paths in vectorially described polygon of genus zero can be performed efficiently (Guibas & Hershberger 1987) as a result of their uniqueness. Discrete 4-connected paths or 8-connected paths are not unique even in genus zero. If we consider polygons with holes, even the continuous case is far more challenging, and efficient solutions seem only possible when the path's homotopy type is given (Bespamyatnikh 2003, Efrat, Kobourov & Lubiw 2002, Hershberger & Snoeyink 1994).

A simple alternative to compute shortest paths in our discrete polygons is breadth-first search using the topological information resulting from the graph that has as nodes the interior cells and edges are present if two interior cells are adjacent (in 4-connectivity, the cells hold a North-South or East-West relationship, while 8-connectivity admits diagonals). Later, we will discuss using geometric information; however, we now present our first algorithm that solves a DMVPP in polynomial time by providing an optimal solution or declaring no solution is possible.

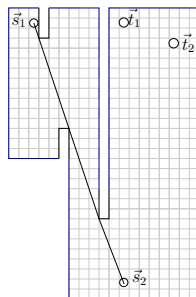


Figure 5: An unfeasible DMVPP. Any movement from the start cells results in visibility loss.

3 An Algorithm for Arbitrary Genus

Definition 6 The region $R_n(c_s, c_t)$ is the union of all discrete paths from c_s to c_t with length n or less. The shortest path region is $R(c_s, c_t) = R_n(c_s, c_t)$ where n is minimal and $R_n \neq \emptyset$.

Definition 7 A minimal successor of a cell c in some shortest path region is a cell connected to c which is the successor of c on some minimal path to the target.

A solution to a DMVPP can be found for any genus by exploring a “feasible states graph”. If the exploration fails, then one learns that this instance of the DMVPP can not be satisfied (as in Figure 5). This is remarkable, since in the continuous case, algorithms are known for genus zero only (ie only for simple vectorial polygons). The “feasible states graph” FSG is constructed as follows. Each node of FSG is made of a pair (c_1, c_2) of interior cells of the polygon that are mutually visible (see Definition 3). A node represents a possible state in the travelling of the agents. An edge between two states (two nodes) (c_1, c_2) and (c'_1, c'_2) is in FSG if c'_1 is a successor of c_1 and c'_2 is successor of c_2 . That is, the agents can move from the state (c_1, c_2) to the state (c'_1, c'_2) . Also, c'_2 does not need to be different from c_2 , so one agent may stay stationary for this step in the schedule.

Clearly that the starting sightline is a state, and thus a node in FSG ; and so is the target sightline. Therefore, any algorithm that finds if these two nodes are connected in FSG determines a feasible solution or declares that the instance is infeasible.

A subgraph of FSG is the graph SMP of states on minimal paths (this graph has the same type of nodes as FSG but an edge between two states (two nodes) (c_1, c_2) and (c'_1, c'_2) is in SMP if c'_1 is a minimal successor of c_1 and c'_2 is minimal successor of c_2). A path from the starting sightline to the target sightline in SMP ensures that it is possible to solve the DMVPP using a minimal path on each agent. This results in an optimal solution of total minimal length. Since Figure 4 shows that failure to find a path in SMP just means that the solution may not be possible with minimal discrete paths, we use FSG with weights to compute a solution of minimal length. The edges of FSG are assigned a weight of 2 if both cells are different in the move represented by the edge while the edge will have a weight of 1 if only one cell moved. Then, a minimal path in the weighted FSG has length equal to the sum of the lengths of the paths of the agents. Our algorithm applied to Figure 4 with FSG weighted will find an optimal solution (although we know it is not made of shortest paths; the mutual visibility constraint forces agents to take detours).

The algorithms for exploring FSG or SMP can be breadth first search, A*, or Dijkstra's algorithm for the shortest path between a source and a target in a weighted graph. The challenge is to construct FSG and/or SMP . To find all nodes we need a means to determine if two cells are visible. If the polygon is converted to a vectorial description or provided as a vectorial polygon, determining the visibility of two points in its interior can be performed efficiently in simple polygons (Guibas, Hershberger, Leven, Sharir & Tarjan 1987). However, if we are given a data structure that rapidly returns if a cell is in the interior or the exterior (for example, in constant time), the visibility between two cells can be determined by testing if any of the cells that intersect the sightline is exterior (and the genus would not impact the complexity).

To find the edges of FSG we only need to test that this node exists (the cells are internal and mutually visible). For a node (c_1, c_2) , there are at most 80 possible adjacent nodes in 8-connected (and 24

in 4-connected) corresponding to the 9 possibilities for an agent made of 8 moves and one for remaining steady. In the worst case, building *FBS* has complexity $O(vis(n)n^2)$ time where $vis(n)$ is the complexity of testing visibility between two cells and n is the number of interior cells (there are $O(n^2)$ nodes and the number of edges is linear in the number of nodes because the degree is bounded by a constant).

The construction of the *SMP* graph requires the computation of the regions $R(c_{s_1}, c_{t_1})$ and $R(c_{s_2}, c_{t_2})$ (alternatively, using a region builder algorithm, the computation of discrete minimal paths μ_1 (from \vec{s}_1 to \vec{t}_1) and μ_2 (from \vec{s}_2 to \vec{t}_2)). Useful data structures are the “successor maps” from cells in $R(c_{s_i}, c_{t_i})$. In these maps, each cell is linked to the set of minimal successors. If “flooding” by breadth-first search to compute μ_1 and μ_2 on a grid representation is used, then $R(c_{s_i}, c_{t_i})$ and the successor maps can be calculated at the same time. The nodes in the graph can be constructed beforehand using knowledge of $R(c_{s_i}, c_{t_i})$ or during exploration starting with (c_{s_1}, c_{s_2}) .

While in the worst case the construction of *SMP* is also $O(vis(n)n^2)$ time, it typically requires $O(vis(n)|R(c_{s_1}, c_{t_1})||R(c_{s_2}, c_{t_2})|)$ time since a hash table implementation of the state map requires $\Omega(|R(c_{s_1}, c_{t_1})||R(c_{s_2}, c_{t_2})|)$ space. It also improves the complexity of the next phase, because now a search for the start sightline to the target sightline is linear in the size of the graph; that is $O(|R(c_{s_1}, c_{t_1})||R(c_{s_2}, c_{t_2})|)$. So the overall complexity is $O(vis(n)|R(c_{s_1}, c_{t_1})||R(c_{s_2}, c_{t_2})|)$ time.

In the next section we attempt to reduce the cost of calculating the regions $R_i(c_{s_i}, c_{t_i})$ for polygons with no holes.

4 Improved efficiency for genus zero cases

In order to reduce the cost for the algorithm in Section 3, we explore conditions that allow us to reduce the cost of finding the shortest path regions. We now develop algorithms for situations where some additional information is known and the polygon has genus zero. First, we study the case where we are provided with a discrete minimal path between the endpoints. We show how the rest of the region $R(c_{s_i}, c_{t_i})$ can be computed in Section 4.1. If such a path is not known, but Π (the Euclidean shortest path between the endpoints) is known, then we show in Section 4.3 that rasterizing Π will give more inexpensive ways to find a discrete minimal path between the endpoints, and thus, by the results to be presented in Section 4.1 an overall faster algorithm.

4.1 Region Builder

If one minimal discrete path is known, the shortest path region $R(c_s, c_t)$ can be calculated efficiently. The worst case complexity of $O(n)$ time, for the number n of internal cells in the polygon, is now $O(|R(c_s, c_t)|)$ time. Again, we assume that there is a constant time test to determine if a particular cell is an external cell or not. The following lemma shows that if we know a cell is in a discrete minimal path and we know one minimal successor, then the options for other minimal successors are limited.

Lemma 1 *Let P be a genus zero 8 or 4-connected discrete polygon. If a minimal discrete path travels in direction α at cell c to cell c^+ , then no other minimal discrete path with the same endpoints travels from c in a direction outside $[\alpha - 90^\circ, \alpha + 90^\circ]$.*

Proof: First the 4-connected case. Without loss of generality suppose that $\alpha = 0$ (rotate P if necessary).

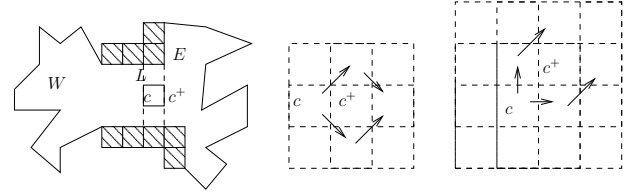


Figure 6: For a cell c and its successor c^+ . (a) If c^+ in a minimal 4-connected path is East; then the target must be in the region E . (b) In 8-connected, if c^+ is East, then only either North-East or South-East can fork other minimal paths (and must be explored, for example the target could be two cells East of c). (c) In 8-connected, if c^+ is North East, then only moves North or East can fork other minimal paths (and must be explored, for example the target could be on North and one North-East move).

Cell c^+ is East of c . Let L be the set of cells (including c) from the vertical chord through c (these are visible to c). The polygon P is now partitioned into three parts L , the “East” part which includes c^+ (denoted E) and the “West” part (denoted W). Refer to Figure 6. We know that $E \cap W = \emptyset$ since the polygon has genus zero, so there is no path from W to E which does not pass through L .

No minimal path through c and c^+ can include any cell in L after c . If it did, then a move North would produce a shorter path. Thus, the target point must be in E . Note that this does not automatically exclude paths which travel North or South from c , since such paths could be postponing the single East move out of L in favour of vertical movement.

Any path which travels West from c will pay a cost of at least 2 additional horizontal moves to get through L . So, such a path cannot be minimal. This establishes the 4-connected case.

In the 8-connected case where $\alpha \in \{\text{North, South, East, West}\}$ the reasoning is the similar but L is defined differently. We let L be all cells with centres on the North-East diagonal and on the South-East diagonal extending from c to the polygon boundary. Cells in L are visible cells. The region E is defined as those cells 4-connected to c^+ , while W is those cells not in $L \cup E$. The target cannot lie in L , since a diagonal sequence would be a shorter path than any path involving c^+ . Similarly, the target cannot be in W since the cell where any path entered W could be reached in fewer steps by using diagonal steps and a vertical movement (or by not retracing steps). So in this case, the lemma is stronger, namely any direction not within 45° of α cannot produce a minimal path.

For the other 8-connected cases, we have $\alpha \notin \{\text{North, South, East, West}\}$. We suppose $\alpha = \text{North-East}$ and rotate the polygon otherwise. Let L be the set of cells with centres on a line through c with gradient -1 and visible to c . The region E is again the cells 4-connected to c^+ while $W = \overline{(L \cup E)}$. As before, if there is a minimal path through c, c^+ , the target cell must be in E . If the target where in W , there would be a cell where a path moving West, South or South-West entered W . But, then this path would be shortened with fewer steps using movements in L .

Figure 6 shows a situation where this “large” range is required. \square

We are now in a position to describe how to obtain $R(c_s, c_t)$ in time proportional to its size using as input a minimal discrete path between c_s and c_t . The algorithm labels internal cells. When cells are labelled as *used* they belong to $R(c_s, c_t)$ while they are labelled as *bad* they are in fact in the boundary of

$R(c_s, c_t)$. Initially, a stack S is loaded with the known minimal path, placing c_t first and working backwards so c_s ends up at the top of the stack. The initialisation also labels these cells as used and marks them with the distance to the target (so the target cell gets distance 0). All used cells c (with the exception of the target) have at least one used cell c^+ that they recognise as their successor.

Repeatedly, the top cell c in S is extracted.

1. **If c is already marked as used**, we use Lemma 1, to find neighbours of c that are possible forks of a new path from c to the target. These neighbours are pushed back into S ; those that represent an angle of larger magnitude with respect to the movement c to c^+ are pushed first and the one with smallest angle in magnitude to the orientation c to c^+ is pushed last. These cells c^+ will not be labelled used nor bad; however, they will be marked with c as the used cell responsible for including them in S .
2. **If the top cell c in S is not a used cell** when extracted from S , then c points to the used cell r that caused its insertion. We use another stack T and insert in T the cell c and all cells in the path starting at c and following the boundary of used cells that is traced from r to the target. The distance values are attached to these cells in T enumerating from the value in r . The cells in the path in T are then classified as used in the region or marked as bad. This analysis proceeds by extracting the top cell c' in T .
 - (a) **If c' has a neighbour cell c_n already used and with distance one less than c'** , (so that c' would reach c_n and become part of a shortest path) then c' is marked as used with its corresponding value and pushed into the stack S ; also, in this case, all of the cells in T are now part of a minimal path, so they are all extracted and as they come out of T , they are inserted in S as used, with the corresponding distance label and successor pointer. Note that in this case, T will be empty and the analysis of the path in it is finished.
 - (b) **If c' has no neighbour cell already used so that c' can reach it and become part of a new minimal shortest path**, then c' is marked as bad and the analysis continues with the next cell in T .

4.2 Complexity and Correctness

Lemma 2 *Let P be a polygon in a 4- or 8-connected discrete polygon of genus zero, given μ a minimal discrete path between c_s and c_t , the algorithm labels a cell as used with its distance to the target if and only if it is in $R(c_s, c_t)$.*

Proof: We show that all cells labelled as used are in $R(c_s, c_t)$ and that no cells in $R(c_s, c_t)$ are missed.

The first part follows because the algorithm labels all cells in μ as used. Or alternatively, it labels all cells in a shortest path from c_s to some cell c , a shortest path from c to c' and a shortest path from c' to c_t . The path c_s, c is shortest because it is so built in S , the path c to c' is shortest because it is so built and stored in T . The path c' to c_t is shortest because we have the shortest distances to c_t recorded. The concatenation of these 3 paths must be a shortest path from c_s to c_t since it has the same length as μ .

The second part is challenging. Since the polygon has genus zero, there can be no holes in $R(c_s, c_t)$.

We proceed by contradiction. Assume there are cells in $R(c_s, c_t)$ which are not marked as used. Then, at least one of them must be a successor to a marked cell because these cells are in minimal paths from c_s to c_t and c_s is marked. We just pick the cell $c \in R(c_s, c_t)$ closest to c_s that was not marked. In order for c not to be marked as used, it must either have not been checked at all or have been marked as bad. Lemma 1 tells us that all possible valid successors for all checked cells are considered. Thus, it must have been marked as bad. Cell c is only marked bad if the path which follows the border of the known region was extended as far as possible and failed to connect anywhere after and including c . The known region would need to double back so it was on both sides of c . This contradicts the statement that μ is minimal. \square

The complexity analysis is as follows. Each cell is introduced into T at most once and into S at most once. All the work is proportional to the push and pop operations in these two stacks. This gives a runtime complexity proportional to the number of cells marked as used and those marked as bad. Since a bad cell is always a neighbour of a used cell, the bad cells are inside the boundary of $R(c_s, c_t)$. Thus, the total work of the algorithm is proportional to $R(c_s, c_t)$. However, for each cell we not only push it or extract it; we also query if its interior or exterior, and we need a data structure per cell to record if it is been labelled (an in that case, the label is used or bad), to record its successor and its distance to the target. We assume that we have a constant time function $\text{IS_INTERIOR}(c) \rightarrow \text{Boolean}$ as part of the given input and we will not charge this algorithm for reading the input and building this function (we may want to run many instances of a DMVPP on the same polygon).

However, we call the “properties storage structure” the data structure for maintaining used/bad labels, successor pointers and distance values for a cell indexed by its integer coordinates in the grid. Under this conditions, the analysis reveals an overall complexity of $O(I + Ak)$ time, where

- I is the cost to initialise the property storage structure,
- A is the number of cells in $R(c_s, c_t)$, and
- k is the complexity of property lookups and border checks for cells.

Since we can find the length $|\mu|$ of the given minimal path in time less than $R(c_s, c_t)$; the identifiers for cells can be normalised to a pair of integers in the range $[0, |\mu|]$. The property storage structure can then be a hash table for which modulus functions would be a “good hash function” (Gonnet & Baeza-Yates 1991). With such a good hash function, a table of size at least A would suffice to retrieve and update properties in $o(1)$ time.

Thus, the overall complexity is $O(A)$ time and space; this is a good result as the complexity is linear in the size of the output. Of course, in the worst case, the complexity can be intimidating because the region can have quadratic size in $|\mu|$. But, for most interesting polygons, this algorithm behaves in complexity much smaller than the number of interior cells of the polygon.

4.3 Covering an Euclidean shortest path

In order to construct the region in the previous section a minimal discrete path is required. In this section we show that, if the Euclidean shortest path Π between cell centres is known, then it can be rasterized to produce such a path. We do this by proving that Π is “covered” by $R(c_{s_i}, c_{t_i})$. While the end result applies to genus zero an intermediate lemma is more general.

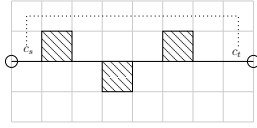


Figure 7: If a 4-connected discrete polygon is not genus zero, then minimum length discrete paths may not cover a continuous line segment. The horizontal segment is between the corners of two cells. Minimum length discrete paths (dotted lines) avoid obstacles, rather than covering the Euclidean shortest path.

Definition 8 A set of cells C in a discrete polygon covers a continuous path π , if $\forall p \in \pi$ implies \exists cell $c \in C$ such that $p \in c$.

Definition 9 The discrete length of a continuous path π , between two cells c_s, c_t in a discrete polygon, is the minimum value k such that $R_k(c_s, c_t)$ covers π .

We now prove that for genus zero discrete polygons, the length of minimal discrete paths is equal the discrete length of the minimal continuous path(Π). We do this by proving that Π is covered by $R(c_s, c_t)$.

Lemma 3 examines individual line segments in Π and shows they are covered then Lemma 4 deals with Π as a whole.

The following lemma establishes the desired result for a single straight line segment. The Euclidean shortest path is made of a polygonal line where the vertices are vertices of the polygon (Guibas & Hershberger 1987) and the line segments are straight. Because the path is from c_s to c_t , the vertices of the Euclidean shortest paths would be corners of cells, or the two centres of the cells c_s and c_t .

Lemma 3 Let P be a discrete polygon which is either 8-connected or (4-connected and genus zero) and Π a Euclidean shortest path between cells c_s and c_t . For each line segment (p_1, p_2) in Π , \exists cells c_1, c_2 such that $p_1 \in c_1, p_2 \in c_2$ and \forall point $p \in (p_1, p_2)$ there is a cell $c \in R(c_1, c_2)$ with $p \in c$.

Note that this lemma establishes the base case for when Π has only one line segment. We employ a standard rasterization algorithm but state it explicitly for clarity.

If Π has more than one line segment, up to three cells could include a segment endpoint (a vertex of Π). We will accept any of these cells for the lemma.

Proof: For simplicity, we assume throughout this proof that the slope of the segment lies between 0° and 90° inclusive, other slopes follow by symmetry.

We first establish the 4-connected case. If both endpoints of the segment are cell centres, a minimum length discrete path can be constructed as follows. Beginning with the start cell c_s , add the current cell to the discrete path. If the segment leaves the cell through either the North edge or the East edge, then add as the next cell the neighbour cell the line enters. Make this new cell the current cell of the discrete path.

If the line segment leaves through the corner, then at least one of the cells to the North or East must be free of obstacles (add either free cell), then add the North-East cell to the discrete path and make it the current cell. Continue until the current cell includes the target point.

This path has minimum length since it has length equal to the Manhattan distance between its endpoints. It also covers the segment.

Two cases remain to establish the 4-connected case. The next case is when at least one of the endpoints of the line segment is a cell corner rather than

a cell centre. The path production algorithm can be modified in the following way. Suppose the start point p lies on a cell corner. We only require a slightly different initialisation. If the vertex p is

- in the North-East corner of c_1 , we add c_1 to the discrete path and either one North or one East cell, followed by the North-East cell.
- the South-East corner of c_1 , we make c_1 the first cell of the discrete path and next in the discrete path is the East cell,
- the South-West corner of c_1 , we start the discrete path with c_1 , and
- the North-West corner in c_1 , we start with c_1 and the North cell.

After this initialisation we continue with the previous algorithm.

Adding extra cells for a non-centred target works in a similar way. Note that the choice of c_1 is not restricted here.

The last case remaining is a horizontal or vertical line segment with endpoints on cell corners (and hence the entire of Π must fall on cell boundaries). In such a case, obstacles can touch the segment. The path can be constructed by adding cells along the line until an obstacle (or the target) forces a change of side. Such a path will cover the segment. It is of minimal length because the only way it could fail to be so is if there was a shorter path around an obstacle which forced a change in side. This will not happen provided the polygon is genus zero. In fact, any line segment which is not vertical, nor horizontal, would be covered by the method above by an optimal discrete 4-connected path, even if the polygon had genus different from zero. But, horizontal or vertical line segments may not be covered by $R(c_s, c_t)$ if the polygon has holes (see Figure 7).

Now, the 8-connected case is slightly more difficult. We will construct a discrete path for Π in the 8-connected case in a similar manner as for the 4-connected case, with two differences. First, we will break each cell into 4 quadrants with a horizontal dividing line through the centre of the cell and a vertical dividing line also through the centre of the cell. These quadrants are named North-West, North-East, South-East and South-West in clockwise order. The second difference is that we will construct a discrete shortest path from the endpoints of the line segment; however, not every point in the line segment will be covered by this discrete path. We will need some further discussion to show that the straight line segment is covered by $R(c_s, c_t)$ by slightly modifying the path built by the method into another shortest path that covers regions not covered by the path before.

The algorithm, however, works again by incrementally building the discrete path guided by the cells intersected by the straight line segment. Recall that we assume the line segment has slope in $[0^\circ, 90^\circ)$ and other cases are handled by rotations and symmetry.

Let c_s be the initial current cell. When analysing the current cell, if the segment leaves the cell via

- the East boundary and in the East cell the segment intersects more than one quadrant of the East cell, move East,
- the North boundary and the straight line intersects more than one quadrant of the North cell, add the North cell as current,
- in all other cases, add the North-East cell as current.

Repeat the analysis of the current cell until the target is reached.

As discussed in (Lovell & Fenwick 2004) the minimum length for a rasterized line segment in 8-connected path is the maximum of the horizontal difference and the vertical difference. Since this method produces a path which is minimal by this measure, it is a minimal 8-connected path.

However, the discrete path produced covers the straight line segment only when such straight line has slope 0° , 45° or 90° . To deal with this, we first assume, without loss of generality, that the segment has slope larger than 0° and smaller than 45° . In this case, the path produced by the method described above will consist only of East and North-East movements and it sometimes moves North-East leaving some part of the straight segment below the path.

Consider the first North-East move which has some part of the segment below it. Following it will be a sequence (possibly empty) of North-East moves terminated by an East move. Since the slope of the segment is less than 45° , a section of the discrete path composed of North-East moves will diverge from the straight line segment until an East move is encountered. This means that all the moves in the sequence leave uncovered parts of the segment below and hence all such cells are free of obstacles. Transposing the first and last moves in the sequence will produce a new path with sequence shifted to the right. This new path covers the points in the segment below the previous path.

If an East movement is followed by a North-East movement, then there may be some part of the segment above the path. In that case, the two movements can be transposed (and the process repeated to handle any other such cases). So the union of all paths produced this way will cover the segment (Note that none of these transformations increases the length of the path).

The same issues arises in the 8-connected case as well. Namely, the line segment may be horizontal or vertical along the grid that defines the cells. Similarly, some careful initialisation may be necessary. For cell c_1 , if the start point is

- South-East, we add the current cell and move East,
- South-West add the current cell,
- North-East add the current cell and move North-east,
- North-West move North.

Once this initial move has been made applied, the method proceeds as before. Similar operations apply for the target cell c_2 . These paths are still of minimum length.

Interestingly, running along cell boundaries does not create the problem in 8-connected as it does in 4-connected (and hence the lemma is true without the genus zero constraint). Because each (navigable) corner must have the three surrounding cells free, it is always possible to switch sides without extra cost, using a diagonal motion. Such a path is of minimal length and covers the segment. \square

The previous lemma tells us about individual straight line segments, now we need to address the whole Euclidean shortest path and show that we cannot do better by taking a different path. Moreover, the lemma provides a linear algorithm in the length of the discrete path.

Lemma 4 Consider a genus zero discrete polygon P , cells c_s, c_t and the Euclidean shortest path Π from c_s to c_t . In such a case Π is covered by $R(c_s, c_t)$.

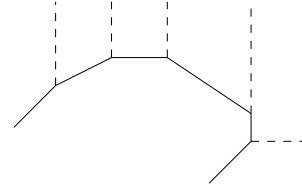


Figure 8: Decomposition of a path as described in Lemma 4.

Proof: Consider constructing the region $R(c_i, c_{i+1})$ for each straight line segment c_i, c_{i+1} of Π as described in Lemma 3. Note that we must be able to match the end-cells around the vertices because 3 cells around an (navigable) corner must be free (Figure 2) and Lemma 3 does not depend on our choice of endpoint. The union of these regions must be a subset of $R_k(c_s, c_t)$ which covers Π for some k . It remains to show that k is minimal.

Since the polygon is genus zero all paths between c_s and c_t must pass the same obstacles. Decompose the polygon as follows, for each segment in Π draw a vertical chord through the endpoint of each segment (these are vertices of the vectorial description of the polygon). If the segment was vertical draw a horizontal chord. See Figure 8.

Now let μ be a discrete path constructed by joining the covers of straight line segments as described in Lemma 3. First we address the 4-connected case. The Euclidean path Π cannot reverse its direction (with respect to the x -axis without a vertical or a horizontal segment). If such segments occur, travelling anywhere other than along Π in that section means the discrete path has paid extra to “go wide” and must pay that cost again to get back in line with Π . So no gains can be made in such sections. For the other sections, take the union of covers for adjacent segments. In such segments, μ will be monotonic with respect to horizontal and vertical directions and have length equal to the Manhattan distance between the endpoint. Hence μ has minimal length.

Now the 8-connected case. In sections for segments of Π with slopes between $\pm 45^\circ$ inclusive, the minimum length of any path through that section is determined by the number of horizontal moves; so μ is minimal. Similarly to 4-connected no savings can be made on a vertical segment (although other paths may be able to match the optimal length). Further, any previously made gains will be lost in such sections.

Provided a sequence of segments is convex, there is no gain to be made by moving away from μ . Non-convex sections arise when there is a segment where the corners which terminate the segment are on “opposite sides”. Any alternate path would need to enter the same point on the section boundary as μ or risk passing on extra cost.

So there are no paths shorter than μ from c_s to c_t , hence k is minimal and Π is covered by $R_k(c_s, c_t) = R(c_s, c_t)$. \square Note again, that we obtain a linear algorithm for finding a discrete path of minimal length guided by the minimum Euclidean path.

5 A Family of Solvable Instances

While we have shown that there are instances of an DMVPP that are not solvable, we show here a large, non-trivial family of 4-connected DMVPPs that admits a solution using minimal paths. The characterization is the following theorem.

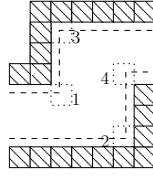


Figure 9: Cells 1, 2, 3 and 4 are examples of complete corners for a 4-connected DMVPP. In this case, cells 2 and 3 will not be processed by our algorithm, since they will be encountered while processing cell 1 and cell 4 respectively.

Theorem 1 Consider a 4-connected DMVPP with non-crossing sightlines where the start cells are aligned (that is, they share either an x -coordinate or a y -coordinate) and the target cells are also aligned. In this case, a schedule exists for any pair of minimal discrete paths between their respective endpoints.

In order to prove this result, we will make use of the following lemma.

Lemma 5 A minimal path in a 4-connected genus zero discrete polygon cannot contain more than one contiguous sequence of cells from any horizontal or vertical chord of cells of the polygon.

Proof: By contradiction. Suppose such a path and chord exist and cells c and d are in the chord and do not belong to the same contiguous sequence of cells in the chord. Since this is 4-connected, moving in the chord from c to d creates an even shorter path. This contradicts that c and d belong to a shortest path. \square

We also will need the following definition.

Definition 10 A complete corner on μ_i is a corner cell c which is aligned both horizontally and vertically with points on the other path. (See Figure 9)

Note that endpoints are also considered corners (but are not necessarily complete).

We now proceed with the proof of Theorem 1. Note that what we need to do is construct a schedule. The proof proceeds by showing an algorithm that builds the schedule in steps. One of these steps is how to navigate a corner. If neither of the agents are at a corner, the algorithm indicates which the agent advances from a vertically aligned position along the optimal discrete paths μ_1 and μ_2 . That is, without loss of generality, we assume the starting sightline is vertical; that is, the cells c_{s_1} and c_{s_2} are vertically aligned (and the horizontal case will follow by symmetry).

The algorithm shows how to move at least one agent, while preserving the invariant that the agents are aligned. For example, if the next step (in the minimal discrete path) of either agent is vertical, then the schedule includes (ie adopts) the move of those agents which have a vertical step. This preserves vertical alignment.

The algorithm will only face the following possibilities, and will choose the action that applies first. That is, the algorithm takes the following as a *decision list* (an option will only be taken if those above it do not apply).

1. One of the agents is at a complete corner.
2. One of the agents' next move is vertical.
3. Both the agents' next moves are horizontal in the same direction.

In fact, the entire proof is based on showing that it is impossible to reach the following situation.

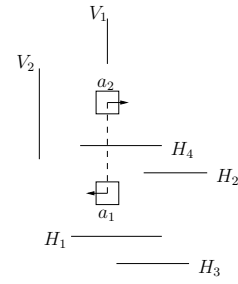


Figure 10: Possible positions for the target sightline T if the agents move in opposite directions.

Definition 11 A scenario where the agents are vertically aligned and both of the agents' next step in a minimal discrete path is horizontal, but in different directions (ie, one West and one East) is called a non-schedule point.

Lemma 6 If the current situation of two agents that have travelled along minimal paths maintaining visibility is that they are aligned vertically, and they are not at a corner, then a non-schedule point is impossible.

Proof: First we show that a non-schedule point (Definition 11) cannot actually occur. Without loss of generality assume that a_1 and a_2 are vertically aligned. Figure 10 shows possible positions for the target sightline T .

- First suppose T is vertical. In this case, we have two further possibilities
 1. In the first, the position (for the target sightline) is to be vertically aligned with the current positions of a_1, a_2 (illustrated as V_1 in Figure 10). This is not possible. If T were visible from a_2 , then the non-vertical moves by the agents cannot possibly be optimal. If T were not visible, then the agents would need to navigate a corner to reach it, and this case would reduce to the next (which we illustrated as V_2).
 2. The target vertical sightline T is to one side of the agents (illustrated as V_2). We argue this is also impossible, because one of the agents (in the figure it would be a_2) moves away from T . In order to reach T , this agent must cross the vertical chord through the agents' initial positions. However, since the polygon is genus zero, the crossing cell would be visible to a_2 . Using Lemma 5, we reach the contradiction that the discrete path for a_2 is optimal.
- Now we analyse the horizontally aligned possibilities for T . This also has several cases.
 1. Consider first when T is as illustrated by H_1 (above or below both agents with an endpoint on either side of the chord through a_1, a_2). One of the agents would need to end at the East end of H_1 . It cannot be a_1 by the same reasoning which ruled out V_2 . Agent a_2 is no good either, since in order to reach H_1 there would need for some later cell on μ_2 which is horizontally aligned with a_1 forming a complete corner. This which is a contradiction. This eliminates H_1 and also H_3 (H_3 is the same as H_1 but without endpoints on both sides of the line through a_1, a_2).

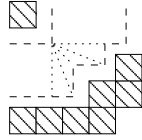


Figure 11: Sightlines in a complete corner.

2. The case illustrated as H_2 is a target sightline whose y coordinate is between a_1, a_2 . This is ruled out by the same reasoning as V_2 .
3. However, ruling out a position like H_4 (a target that is crossing the sightline between a_1, a_2) takes a little more work. Notice that we cannot have a_1 heading for the East end since this would make μ_1 non-minimal. Further, a_1 and a_2 cannot be at their starting positions, since we do not allow the initial and target sightlines to cross.

So where were the agents before now? Being further apart vertically but still on opposite sides of H_4 would not solve the problem of the initial sightlines crossing. If they started on the same side of H_4 , then the chord through H_4 and Lemma 5 contradicts the minimality of its path.

What is the orientation of S (the start sightline for the problem)? A horizontal S would have required at least one of the agents (say a_i) to be level horizontally with and visible to T in the past. But this means that μ_i cannot be optimal (Lemma 5).

For a vertical S one of three problems occurs.

- (a) S is vertically aligned with and visible to a_2 . In such a case one of the agents will intersect the chord through H_4 more than once.
- (b) Whichever side of the chord through a_1, a_2 the sightline S is on, one of the agents is moving back in that horizontal direction. If an obstacle did not force this reversal of direction, then at least one of the paths cannot be minimal. If it was forced by an obstacle, then both agents would have been on the same side of the chord through H_4 reducing to item 3a

□

We now complete the algorithm. Here is the decision list and its actions.

1. When one of the agents is at a complete corner and the two agents are vertically aligned, use symmetry or rename the agents so a_1 is on the corner. Navigate the corner from vertical alignment to horizontal alignment as follows. Agent a_1 waits while a_2 moves until it is next horizontally aligned with a_1 . We use Figure 11 to illustrate that visibility is preserved during these moves. No obstacles can be inside the region bounded by μ_2 and the sightlines when the agents are aligned, so the only way for any of the sightlines to be blocked is if μ_2 reversed direction and moved behind an obstacle. Lemma 5 excludes such a possibility.

Both agents are never at a complete corner simultaneously (like that in Figure 12) because then at least one of the paths cannot be minimal. Since the polygon has genus zero, there is no reason to turn away from the direct path.

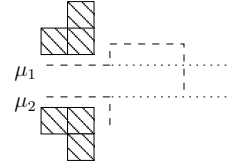


Figure 12: Why a schedule never needs to complete corners starting at the same time. In order to produce a complete corner μ_1 must move north. In order to align with μ_2 's corner it must travel South since the polygon is genus zero this makes μ_1 non-minimal.

2. When one of the agents' next move is vertical the schedule adopts the relevant agents' vertical move.
3. If both agents' moves are horizontal in the same direction, then the schedule adopts both moves.

For space reasons we do not show here that it is impossible for alignment to change from vertical to horizontal in a single move unless the agents are occupying the same cell are some point and then how we deal with agents occupying the same cell.

It is remarkable that DMVPPs with aligned sightlines can be solved so efficiently (compared to our previous algorithms), since we only need to find minimal discrete paths for the agents and then the schedule can essentially be found in linear time. The next subsection shows that this also has implications for the generic instances.

5.1 Impact on All Instances

One approach to deal with problems where the endpoints are not aligned is to decompose the problem into (up to three) sections. For example, find the first cell on μ_2 which is aligned with c_{s1} . Denote this cell s'_2 . Hence if a schedule exists to move a_2 from c_{s2} to s'_2 (and possibly another schedule to handle the target cells similarly), we have a complete schedule since Theorem 1 guarantees us a schedule if the endpoints are aligned.

Lemma 7 *Consider a DMVPP where the start cells are not aligned with agents a_i and a_j . Further, suppose $\exists s'_i \in \mu_i$ such that s'_i is (vertically or horizontally) aligned with c_{sj} and s'_i is the earliest such point. In such a case, the subproblem of moving a_i from c_{si} to s'_i along some minimal path and a_j stationary has one of two outcomes: there is a schedule which uses a minimal path from c_{si} to s'_i or there is no schedule at all for any path.*

Proof: If a schedule exists for the some minimal path from c_{si} to s'_i , then we are done. We prove the alternative case where no minimal path will do by contradiction. Suppose that no minimal path exists from c_{si} to s'_i which has a schedule but there exists a longer path that does have a schedule. See Figure 13. Assume that there is no minimal solution for which a schedule exists. Let c be a cell with maximal distance along some minimal path such that there is a schedule to move a_i from c_{si} to c without losing visibility of a_j and no further movement along any minimal path is possible.

Without loss of generality assume that S , the cell to the south of c is a minimal successor. Note that visibility of this cell must be blocked. Now consider the possible positions of c_{sj} . It cannot be directly south of c since a blockage of S implies that c is not visible either. It cannot be directly East of c since then

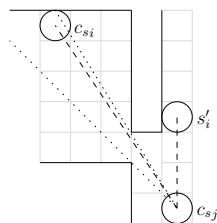


Figure 13: Part of a DMVPP before the start cells are aligned. Dashed lines are sightlines, dotted lines show the visible area between the two obstacles.

S cannot be a minimal successor (this also excludes North). What about somewhere between South and East? Since visibility of S is obstructed and the polygon is genus zero, no path can travel through or below S and retain visibility. If the cell E East of c is a border cell, then there can be no possible path. Alternatively E must be a minimal successor, which must be blocked (else c is not maximal distance along). The same reasoning as for S prevents a path moving past E , so there is no possible path. So we have our contradiction. The other cases follow by symmetry. \square

6 Final Remarks

We have produced very effective algorithms for the discrete version of two agents travelling inside a polygon of arbitrary genus and maintaining visibility while minimising the distance travelled. It is not hard to see that our algorithm generalises for even more agents. If k agents are travelling, the *FSG* graph would be made of k -tuples indicating the current cell of each agent. The *SMP* graph would have nodes if the cells have the desired visibility constraint (for example, a spanning tree of visibility or the complete visibility between the agents).

Also, we have produced algorithms to rapidly construct this graph from the heuristic provided by the Euclidean shortest path in a vectorial polygon. In the case of genus zero, we have shown that the heuristic actually produces the necessary minimal discrete paths. We also have found that aligned starting and ending positions allows very efficient solutions in the 4-connected case. Since the current solution for the continuous case are not as comprehensive, we will explore how effective a discrete solution is as an heuristic for a continuous case if one were to rasterize a vectorially described polygon.

References

- Bespamyatnikh, S. (2003), 'Computing homotopic shortest paths in the plane', *Journal of Algorithms* **49**(2), 284–303.
- Coeurjolly, D., Miguet, S. & Tougne, L. (2004), '2D and 3D visibility in discrete geometry: an application to discrete geodesic paths', *Pattern Recognition Letters* **25**(5), 561–570.
- Davis, I. (2000), Warp speed: Path planning for star trek: Armada, in 'AAAI Spring Symposium on AI and Interactive Entertainment.', AAAI Press, Menlo Park, CA.
URL: www.qrg.cs.northwestern.edu/atgames.org/2000papers.html
- de Berg, M., van Kreveld, M., Overmars, M. & Schwarzkopf, O. (2000), *Computational Geometry Algorithms and Applications*, second edn, Springer Verlag, Berlin.
- Efrat, S., Kobourov, S. & Lubiw, A. (2002), Computing homotopic shortest paths efficiently, in 'Proceedings of the 10th Annual European Symposium on Algorithms', pp. 411–423.
- Fenwick, J. & Estivill-Castro, V. (2005), Optimal paths for mutually visible agents, in X. Deng & D.-Z. Du, eds, 'Algorithms and Computation, 16th International Symposium ISAAC', Lecture Notes in Computer Science 3827 Springer, Sanya, Hainan, China, pp. 869–881.
- Fink, E. & Wood, D. (2003), 'Planar strong visibility', *International Journal of Computational Geometry and Applications* **13**(2), 173–187.
- Flocchini, P., Prencipe, G., Santoro, N. & Widmayer, P. (2005), 'Gathering of asynchronous robots with limited visibility', *Theoretical Computer Science* **337**(1-3), 147–168.
- Gonnet, G. & Baeza-Yates, R. (1991), *Handbook of Algorithms and Data Structures, 2nd edition.*, Addison-Wesley Publishing Co., Don Mills, Ontario.
- Guibas, L., Hershberger, J., Leven, D., Sharir, M. & Tarjan, R. (1987), 'Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons', *Algorithmica* **2**, 209–233.
- Guibas, L. J. & Hershberger, J. (1987), Optimal shortest path queries in a simple polygon, in 'SCG '87: Proceedings of the third annual symposium on Computational geometry', ACM Press, New York, NY, USA, pp. 50–63.
- Hershberger, J. & Snoeyink, J. (1994), 'Computing minimum length paths of a given homotopy class', *Computational Geometry: Theory and Applications* **4**(2), 63–97.
- Ickling, C. & Klein, R. (1992), 'The two guards problem', *International Journal of Computational Geometry and Applications* **2**(3), 257–285.
- LaValle, S. (2006), *Planning Algorithms*, Cambridge University Press, UK.
- Lovell, N. & Fenwick, J. (2004), Linear time construction of vectorial object boundaries, in M. Hamza, ed., 'IASTED Signal and Image Processing Conference', ACTA Press, pp. 914–919.
- O'Rourke, J. (1987), *Art Gallery Theorems and Algorithms*, Oxford University Press, New York.
- O'Rourke, J. (1994), *Computational Geometry in C*, Cambridge University Press, UK.
- Preparata, F. & M.I., S. (1985), *Computational Geometry An Introduction*, Texts and Monographs in Computer Science, Springer-Verlag, New York.
- Sack, J. & Urrutia, J., eds (2000), *Handbook of Computational Geometry*, Elsevier.
- Schuijser, S. & Wood, D. (1999), 'Multiple guard kernels of simple polygons', *Journal of Geometry* **66**, 161–168.
- Widmayer, P., Wu, Y., Schlag, M. & Wong, C. (1986), 'On some union and intersection problems for polygons with fixed orientation', *Computing* **36**, 183–197.

Exploring Human Judgement of Digital Imagery

Timo Volkmer

James A. Thom

S.M.M. Tahaghoghi

School of Computer Science and Information Technology

RMIT University

PO Box 2476V, Melbourne, Australia 3001

Email: {tvolkmer,jat,saied}@cs.rmit.edu.au

Abstract

Statistical learning methods are commonly applied in content-based video and image retrieval. Such methods require a large number of examples which are usually obtained through a manual annotation process, that is human raters review images and assign semantic concept labels. The human judgement, however, cannot be regarded as the ultimate truth because of its subjectiveness and the likelihood of human error. We can address these issues by using multiple judgements per example, but evaluating and resolving disagreement between raters is problematic. Moreover, the nature of rater disagreement and how to minimise it are not yet well explored. In this paper we present results of a user study that was specifically designed to investigate human judgement of digital imagery. We discuss the influence of factors such as size and type of semantic vocabulary on inter-rater agreement. We demonstrate the application of latent class analysis for combining multiple judgements. Known from applications in the medical and social sciences, this statistic allows robust, quantitative evaluation of multiple judgements per subject. We believe it is well suited for application during the evaluation and modelling phase in semantic image and video retrieval.

Keywords: image annotation, semantic annotation, latent class analysis, inter-rater agreement

1 Introduction

Supervised learning methods such as Bayesian networks or support vector machines are widely used for automatic image and video classification in multimedia information retrieval. Based on positive and negative training examples, such systems compute a binary model which can be used to classify unseen data automatically. The examples for training have to be annotated with the semantic concept labels taken from a pre-defined vocabulary. The annotation task — performed manually by human reviewers — is time consuming and prone to errors. Moreover, the annotation is often based on only an individual opinion, that is only one judgement per image is obtained from a single rater. In this case, it is impossible to quantify the quality of the annotation unless another individual reviews all annotations.

The effectiveness of automatic classifiers depends on the underlying model, the quality of which in turn depends heavily on the training collection. This dependency is both qualitative and quantitative in

nature. Automatic classifiers need a large number of positive examples per concept but also require the annotation to be accurate and discriminative. Based on this need for large training collections, we want to minimise both annotation error and annotation time when generating annotated collections.

Given enough resources, we can generate multiple annotations per image by assigning several raters to the task. We expect an improvement of the annotation quality because we can model the view of several raters rather than only an individual opinion. However, we often observe significant disagreement between judgements. In the absence of a “gold standard” it is problematic to decide whether an image should be used as a positive or as a negative example. It is often impossible to classify an image into a category with 100% certainty as there is almost always room for ambiguity. Reverting to only using images that have unanimously been classified is a possible solution if the collection is large enough. But for rare concepts, or in smaller collections, this is likely to result in too few positive examples.

Some examples should perhaps neither be used as a negative nor as a positive example if there is much disagreement in their annotation. One might also decide to review the annotation in case the inter-rater agreement is below an acceptable level. Knowing that we almost always have to expect some disagreement when working with multiple judgements, we can quantify the annotation quality based on the level of inter-rater agreement. It is desirable to apply a robust statistic that supports both classifying annotated examples and estimating the rater agreement as a measure of annotation quality.

Driven by the above observations, we want to learn which factors influence the agreement between multiple human raters. We want to clearly classify individual images based on multiple judgements, and finally, we want to minimise the annotation time per image.

In this paper we present results of a user study addressing the reliability and efficiency issues of image annotation. As the problem of video retrieval is usually reduced to an image retrieval problem, we believe this work is applicable to both areas. We have explored different modes of annotation, and used different semantic concepts to study their impact on annotation quality. As a novelty in this field, we apply Latent Class Analysis (LCA) to compute concept prevalence and classification error, and compare it to the inter-rater agreement using intraclass correlation.

After presenting related research in the following section, we describe our experiment in Section 3, followed by an analysis of the results in Section 4. We conclude the paper with a discussion of the lessons learnt and an overview of our future work in Section 5.



Figure 1: Screenshot of the annotation system that we used. In multiple-concept mode all five concepts are activated, as shown here, while only one concept is activated for selection in single-concept mode.

2 Related Work

In prior work with others (Volkmer et al. 2005) we investigated some of the issues related to manual image and video annotation. Within the TRECVID¹ 2005 Annotation Forum, 61,904 key frames were collaboratively annotated by more than 102 reviewers. We used this opportunity to analyse the annotation including additional information in regards to timing and preferences of the input device. This gave us valuable clues, but it was difficult to reliably quantify concept frequencies and inter-rater agreement because the data was collected in an uncontrolled environment. For example, we were not able to determine whether annotations under one user account were indeed done by a single user. Moreover, not all images were reviewed by the same number of annotators. We hypothesised that the agreement between raters decreases if the judges try to annotate too many concepts simultaneously. Based on the data available, we were not able to confirm this. Many questions remained unanswered which led us to conduct a new experiment that is the subject of this paper.

Given the recent trend to share information on the Internet in blogs and online photo albums, we can find many examples in which large image collections are annotated. Most of them, however, do not use a controlled concept lexicon, and instead work with free-text annotations. Perhaps the most prominent example of such an approach is the Yahoo! photo sharing portal Flickr.² Each image can be assigned one or more tags describing some semantics of the image content. This annotation is highly subjective which makes a Flickr image collection less suitable for learning a specific concept. For example, the search term “Airplane” retrieves many images of people sitting in passenger aircraft or aerial views out of airplanes. These noisy results seem to occur because annotations are based on subjective cogitations that individuals connect with an image. The image con-

tent is not necessarily well reflected by the chosen terms. The ESP Game (von Ahn & Dabbish 2004) combines annotations by multiple users in a game-like application. Users annotate images that are found on the Internet; the system computes a confidence score based on how many different users agree on a certain description of an image. Users in turn collect points if they agree on an annotation. While this approach is very interesting, the annotation data is not available to us and we are unable to make a statement about its quality or usefulness for information retrieval purposes.

Other than the studies conducted in connection with the TRECVID Annotation Forums in 2003 and 2005 (Lin et al. 2003, Volkmer et al. 2005), we are currently unaware of any related studies for image annotation that use controlled concept vocabularies. However, analysis of survey reliability and response agreement is a well-studied area in the medical, behavioural, and social sciences. Latent Class Analysis (Lazarsfeld & Henry 1968) is an established method to evaluate survey data. Uebersax & Grove (1990) demonstrate its usefulness for analysing medical diagnostics agreement and extend this work beyond the medical sciences (Uebersax 1992, Uebersax & Grove 1993). The flexibility and robustness of latent class analysis has been demonstrated in other applications (Vermunt 1996, Vermunt & Magidson 2003, Vermunt 2003).

Deerwester et al. (1990) and Dumais (1995) use latent class analysis for an approach for indexing text documents and report improvements in retrieval performance over traditional term matching techniques. Hofmann (1999) extends this approach and proposes Probabilistic Latent Semantic Indexing (PLSI) for text document indexing. The idea underlying these indexing approaches is that the topic of a text document can be expressed as a latent variable that can only be indirectly obtained through modelling document terms as a response vector. In contrast to this work, we apply latent class analysis in evaluation and modelling of multiple judgements in annotated image collections.

¹<http://www-nlpir.nist.gov/projects/trecvid>

²<http://www.flickr.com>

Concept set a <i>vehicles</i>		Concept set b <i>settings/scenes/sites</i>	
a_1 Car	6%	b_1 Outdoor	33%
a_2 Truck	2%	b_2 Sky	13%
a_3 Bus	< 1%	b_3 Studio	11%
a_4 Airplane	1%	b_4 Building	12%
a_5 Boat/Ship	< 1%	b_5 Vegetation	10%

Table 1: The two sets of concepts, $a = \{a_1, \dots, a_5\}$ and $b = \{b_1, \dots, b_5\}$, that we have used in our experiment along with the expected prevalences based on prior experiments.

In the next section, we will describe our experiment before we discuss our findings in Section 4. We apply latent class analysis to compute concept prevalences and outline the possible usefulness of this statistic in the further process of semantic indexing of image collections.

3 Our Experiment

We designed the experiment to study human judgement of images in a controlled environment. We did not aim to generate an annotated collection of examples for training. Figure 1 shows a screenshot of the web-based application that we used. It was specifically designed for this task and allowed users to annotate one image at a time with either a single or multiple concepts.

To be able to relate our conclusions back to prior work (Volkmer et al. 2005), we selected images and semantic concepts for the annotation from the collection that was used in the TRECVID 2005 Annotation Forum. Based on the data collected in 2005, we were able estimate the expected prevalences for the concepts that we use, as shown in Table 1. We describe the concept definitions and the image collection in detail below.

3.1 Methodology

We invited 20 users, 10 male and 10 female, to participate as annotators in our experiment. The participants were drawn from the general public and a pool of research students from the Science, Engineering, and Technology Portfolio at RMIT University. All users were familiar with the use of computers but mostly not experienced in the field of multimedia information retrieval. The experiment was conducted anonymously, that is each user was randomly assigned an anonymous user account so that responses could not be traced back to individual participants.

We allowed a brief training phase so that the participants could get used to the system before we started the experiment. All users were presented one image at a time and (depending on the annotation mode) either one concept or five concepts next to the image. The task asked to select all concepts that a user considered applicable to the image. The annotation system offers navigation buttons for users to go backwards and forward between images. However, we divided the image collection into several sets as described below. Free navigation forward and backwards was only allowed within one image set. We introduced this restriction to provide a minimum of guidance through the collection while allowing users to correct any errors they might make. Each user performed the experiment in two parts; one where they annotated multiple sets of images, each with a single concept that is varied between sets, and one where

Group	User	Part 1	Part 2
1	user1	a1A1, a2A2, a3A3, a4A4, a5A5	b1 b2 b3 b4 b5 B1
	user2	a1A2, a2A3, a3A4, a4A5, a5A1	b1 b2 b3 b4 b5 B2
	user3	a1A3, a2A4, a3A5, a4A1, a5A2	b1 b2 b3 b4 b5 B3
	user4	a1A4, a2A5, a3A1, a4A2, a5A3	b1 b2 b3 b4 b5 B4
	user5	a1A5, a2A1, a3A2, a4A3, a5A4	b1 b2 b3 b4 b5 B5
2	user6	b1 b2 b3 b4 b5 B1	a1A1, a2A2, a3A3, a4A4, a5A5
	user7	b1 b2 b3 b4 b5 B2	a1A2, a2A3, a3A4, a4A5, a5A1
	user8	b1 b2 b3 b4 b5 B3	a1A3, a2A4, a3A5, a4A1, a5A2
	user9	b1 b2 b3 b4 b5 B4	a1A4, a2A5, a3A1, a4A2, a5A3
	user10	b1 b2 b3 b4 b5 B5	a1A5, a2A1, a3A2, a4A3, a5A4
3	user11	a1 a2 a3 a4 a5 A1	b1B1, b2B2, b3B3, b4B4, b5B5
	user12	a1 a2 a3 a4 a5 A2	b1B2, b2B3, b3B4, b4B5, b5B1
	user13	a1 a2 a3 a4 a5 A3	b1B3, b2B4, b3B5, b4B1, b5B2
	user14	a1 a2 a3 a4 a5 A4	b1B4, b2B5, b3B1, b4B2, b5B3
	user15	a1 a2 a3 a4 a5 A5	b1B5, b2B1, b3B2, b4B3, b5B4
4	user16	b1B1, b2B2, b3B3, b4B4, b5B5	a1 a2 a3 a4 a5 A1
	user17	b1B2, b2B3, b3B4, b4B5, b5B1	a1 a2 a3 a4 a5 A2
	user18	b1B3, b2B4, b3B5, b4B1, b5B2	a1 a2 a3 a4 a5 A3
	user19	b1B4, b2B5, b3B1, b4B2, b5B3	a1 a2 a3 a4 a5 A4
	user20	b1B5, b2B1, b3B2, b4B3, b5B4	a1 a2 a3 a4 a5 A5

Figure 2: The specification of our experiment, each user annotated 360 images. For each image/concept pair, we obtain 4 ratings. Lowercase letters represent concepts, while uppercase letters each represent a set of 60 images.

they annotated one set of images with multiple concepts. For a given user, neither the images nor the concepts overlapped between parts.

The primary goal of our experiment was to study the effects on efficiency and inter-rater agreement of the following two factors:

- Concept vocabulary: Specific objects, such as *Car* or *Airplane*, that are less prevalent versus different shot-settings with higher prevalence, for example *Vegetation* or *Sky*.
- Annotation mode: Annotating one image in regards to a single concept at a time versus annotating one image with five concepts simultaneously.

During the experiment, we recorded the annotation generated by each user in a central database. Additionally, we measured the time that each user spent per image. After the experiment we asked users to complete a short survey with simple questions in regards to the annotation.

3.2 Specification

We randomly selected 600 images out of the TRECVID 2005 development collection (Over et al. 2005) for our experiment. This collection contains 61,904 key frames extracted from from U.S. American, Arabic, and Chinese television recordings. These recordings consist mostly of news programs interrupted by advertisement segments and some entertainment programs.

We divided the image collection into two equal parts, A and B , each comprising 300 images. Each set was then further divided into five subsets of 60 images each $A = \{A_1, \dots, A_5\}$, and $B = \{B_1, \dots, B_5\}$.

From the 44 concepts that Naphade et al. (2005) had prepositioned for the 2005 Annotation Forum for TRECVID, only 39 were finally used. We selected ten concepts out of these 39 and defined two

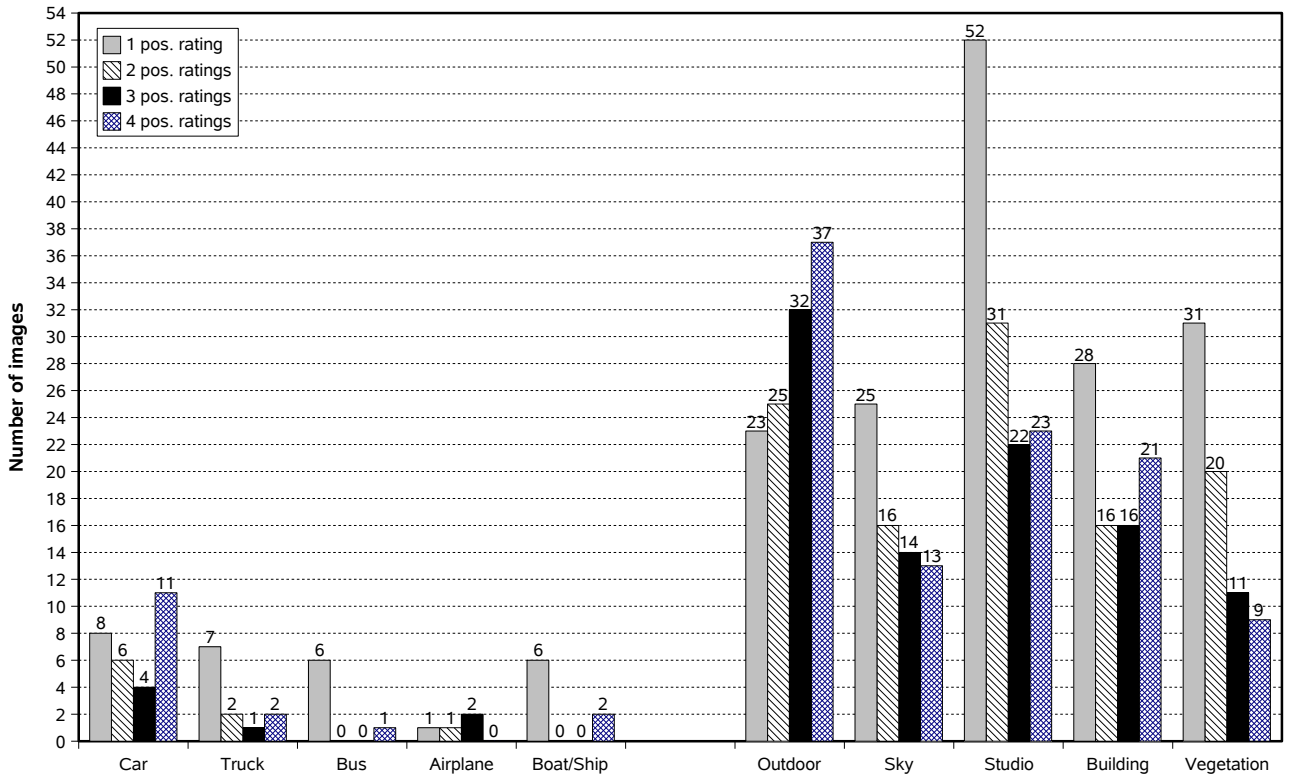


Figure 3: The raw positive ratings for each concept. Each bar represents the number of images that received 1, 2, 3, and 4 positive ratings for the respective concept. We observe substantial disagreement in the ratings.

sets a and b . These were selected such that one set $a = \{a_1, \dots, a_5\}$ represents well-known objects that appear rather rarely in the collection, while the second set $b = \{b_1, \dots, b_5\}$ consisted of settings and scenes with a significantly higher prevalence, as can be seen from Table 1.

Our experiment setup is illustrated in Figure 2. We grouped the 20 participants into four groups of five users each, and paid attention to achieving a uniform distribution of male and female participants among all groups. In Figure 2, a concept/image set combination such as a_1A_3 means that the user annotates the image set A_3 with concept a_1 (Car). Analogue to this, $b_1b_2b_3b_4b_5B_3$ means that the user annotates image set B_3 with all five concepts of concept group b . Each user would therefore annotate six subsets of images, that is 360 images, with all available concepts. As described above, the experiment was divided into two parts so that each user was to use both annotation modes. In single-concept mode, users were to annotate five subsets of 60 images each with one concept. The concept was varied for each subset. In multiple-concept mode, users were required to annotate one subset — 60 images — with five concepts, but this time with all five concepts simultaneously. To cancel out unwanted effects, we rotated the order in which users would see images among all users, and we rotated the order of annotation modes among user groups.

We believe this setup allows us to evaluate concept prevalences and inter-rater agreement based on the different annotation modes as well as in combination. Moreover, we can draw conclusions about which mode might be faster and we can make a statement about the impact of different concept types on the annotation.

In the next section, we discuss the results of our experiments after evaluating the data with latent class analysis. We conclude the paper summarising the lessons we have learnt in Section 5.

4 Evaluation of Experimental Results

The experiment setup results in four independent ratings per image, two for each single-concept and multiple-concept annotation mode. As not all users have annotated all 600 images with all available concepts, we can evaluate 300 distinct images for each concept based on four ratings.

Based on our prior experience, we anticipated the annotation speed to be faster if users have to annotate only one concept as opposed to five concepts at a time. We also expected the agreement between raters to be higher when annotating only a single concept. In addition to these effects, we expected users to have greater difficulty annotating the second set of concepts, set b , because these leave more room for ambiguity. We thus expected a higher disagreement rate for these concepts.

When evaluating the annotation, we are primarily interested in the positive ratings, that is the images that can be used as positive examples for training. All other images are usually considered negative examples. In our case, each image can obtain a maximum of four positive ratings. An optimal annotation would consist only of images with either zero or four positive ratings.

We counted the raw judgements, grouped by the number of positive ratings per image, and visualise these in Figure 3. Each concept is represented by four bars. Each bar shows the number of images that were judged as a positive example by 1, 2, 3, or 4 raters, respectively.

The graph in Figure 3 illustrates well the primary difficulty when using multiple ratings: there is significant disagreement in the number of positive judgements of most images. Consider concept *Airplane* in Figure 3 as an example. There is one image that one user has labelled as depicting an airplane. For another image, two raters have agreed that the image shows an airplane, however, two raters still disagreed

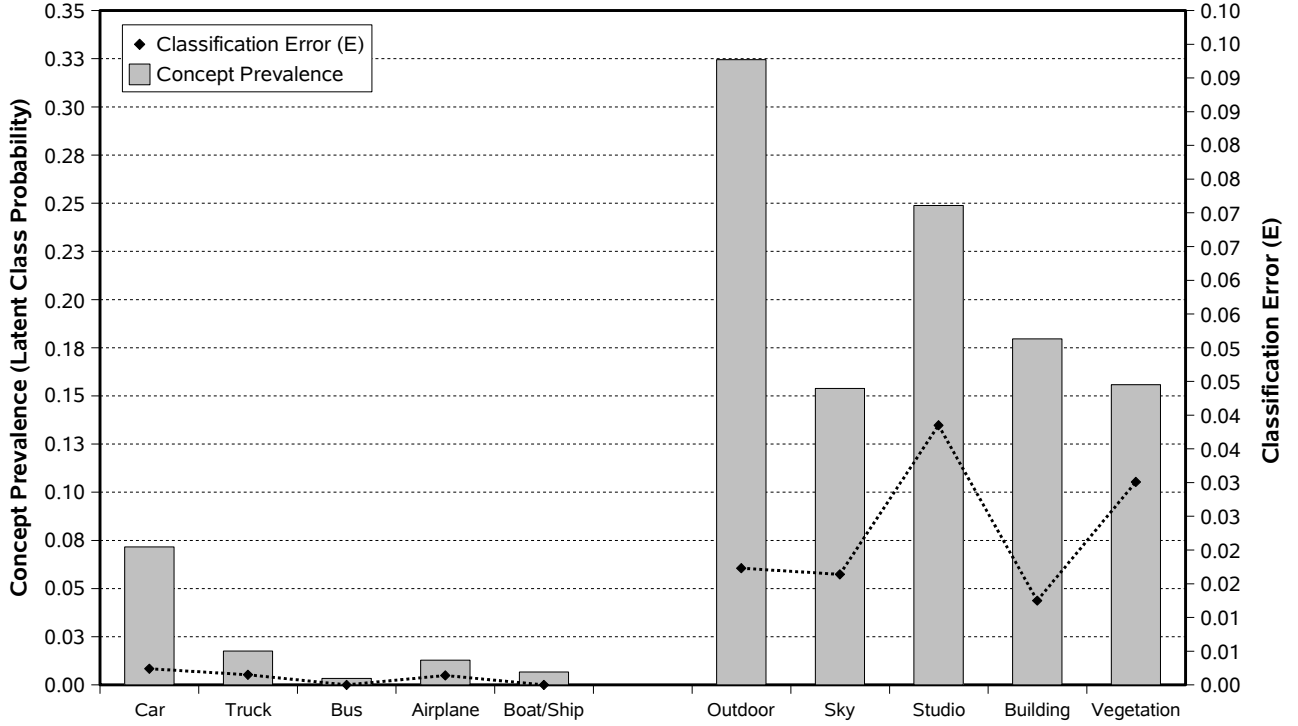


Figure 4: Concept prevalences, calculated using LCM, along with the estimated classification error E . We observe a generally higher error for the more frequent concepts of concept category b . E reaches 0 for concepts *Bus* and *Boat/Ship* because of the few positive examples for these.

with that. There are only two images that a majority of three raters labelled as depicting an airplane. And no single image was unanimously judged by all four raters as showing an airplane.

4.1 Latent Class Modelling

To solve this problem, we apply latent class analysis, or Latent Class Modelling (LCM) (Lazarsfeld & Henry 1968, Vermunt & Magidson 2003). This statistic allows us to combine multiple ratings per image and express these as a probability P that the respective image is indeed either positive or negative in regards to a particular semantic concept.

We adopt the basic idea of latent class modelling that the real classification of an image is contained in our annotation only as a latent variable X . This means that we cannot directly observe it, but we can derive it from the four observable ratings that we have for each image. We combine these ratings in the response vector \mathbf{Y} . According to the latent class modelling approach (Lazarsfeld & Henry 1968, Vermunt & Magidson 2003), the likelihood of obtaining a particular response pattern $P(\mathbf{Y} = \mathbf{y})$ for one item can be described as follows:

$$P(\mathbf{Y} = \mathbf{y}) = \sum_{x=0}^{C-1} P(X = x)P(\mathbf{Y} = \mathbf{y}|X = x) \quad (1)$$

Where C denotes the number of latent classes. We apply a restricted dichotomous model that assumes the two latent classes $x = 0$ (negative) and $x = 1$ (positive), expressed in the dichotomous latent variable X . $P(X = x)$ is the proportion of images belonging to the specific latent class x . This latent class model assumes mutual independence between all ratings for a latent class. The restriction that we apply is interchangeability of raters. This means that ratings of a particular image/concept pair can originate from

Concept set a		Concept set b	
<i>vehicles</i>		<i>settings/scenes/sites</i>	
a_1 Car	7.16%	b_1 Outdoor	32.45%
a_2 Truck	1.76%	b_2 Sky	15.39%
a_3 Bus	0.33%	b_3 Studio	24.88%
a_4 Airplane	1.28%	b_4 Building	17.96%
a_5 Boat/Ship	0.67%	b_5 Vegetation	15.58%

Table 2: Concept prevalences computed based on the new annotation obtained, using our two-class latent class model.

any of the 20 raters. The unrestricted model would consider all ratings per image to always be from the same raters, but this is not the case in our experiment.

To assign an individual image to a particular class, we apply the following Bayesian rule:

$$P(X = x|\mathbf{Y} = \mathbf{y}) = \frac{P(X = x)P(\mathbf{Y} = \mathbf{y}|X = x)}{P(\mathbf{Y} = \mathbf{y})} \quad (2)$$

We use modal classification, that is an image would be assigned to the class for which the highest probability $P(X = x | \mathbf{Y} = \mathbf{y})$ is computed.

Using the ℓEM (Vermunt 1997) program, we assessed our annotation with the two-class model described above. We computed the latent class membership probabilities for all concepts as values between 0 and 1. These can be interpreted as the likelihood of obtaining a positive example for the respective concept if one picked a random image out of our collection. The results are shown in Table 2, expressed as percentage values for each concept. For example, we observe 7.16% of the collaboratively judged images are rated as *Car*, that is approximately 21 images out of 300 depict one or more cars.

The results obtained through latent class modelling correlate well with the predicted concept preva-

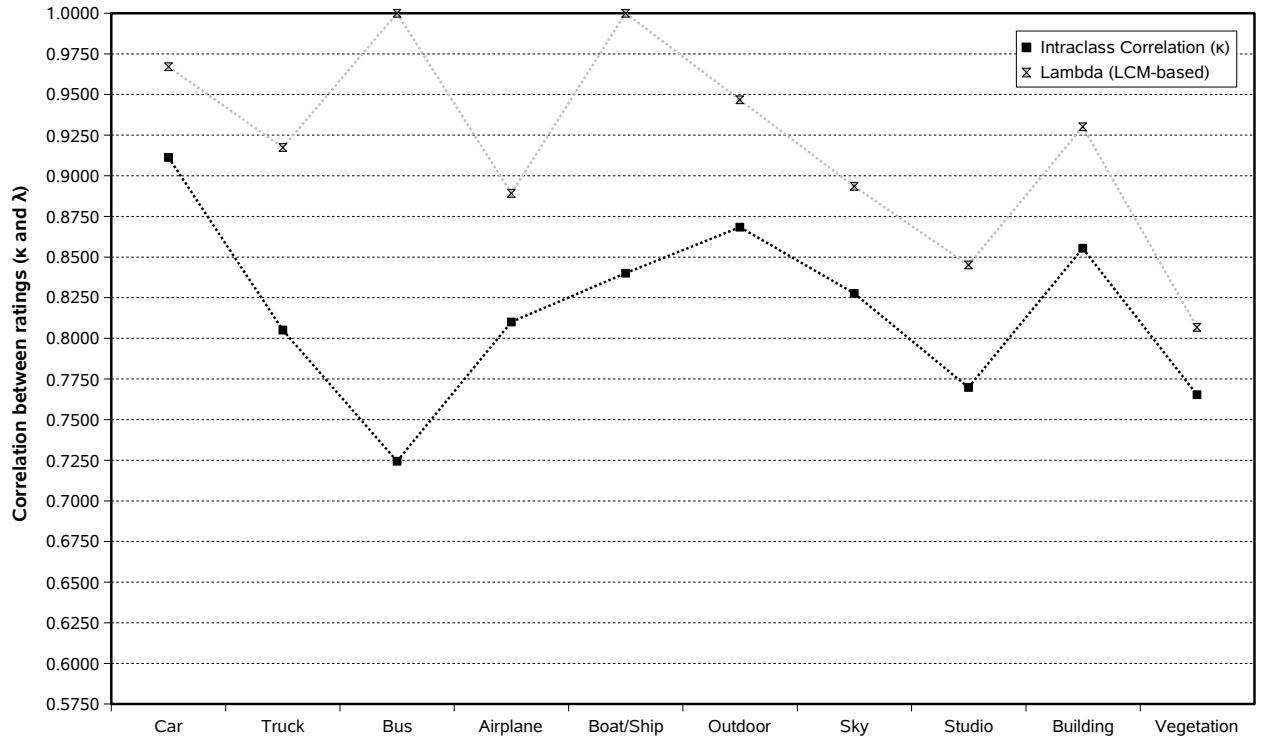


Figure 5: LCM classification performance λ compared to the intraclass correlation index κ . We observe a strong correlation between both: images can be better classified if there is less disagreement in how they have been rated. The concepts *Bus* and *Boat/Ship* are interesting outliers.

lences in Table 1. However, we believe the results computed with latent class modelling are more reliable, because our earlier results were only estimates based on a simple algorithm that counted the positive ratings for each image.

4.2 Estimating Classification Performance

An important measure for the classification performance of a model is the classification error E . It is the estimated proportion of false classifications based on all classifications for a particular concept. When using the modal classification rule as described in Equation (2), we estimate the classification error E for N images rated as follows:

$$E = \sum_{i=1}^I \frac{n_i}{N} \{1 - \max[P(X = x | \mathbf{Y} = y_i)]\} \quad (3)$$

Where I denotes the possible number of different response patterns and n_i the observed frequency for a particular response pattern.

Not surprisingly, the classification error has a direct relationship to the rater agreement because an image can be classified better if more raters agree. Figure 4 shows the computed concept prevalences graphed for each concept along with the classification error. In the case of the concepts *Bus* and *Boat/Ship*, the classification error is estimated to be 0. An explanation for this effect is found when examining the raw positive ratings shown in Figure 3. For the concepts *Bus* and *Boat/Ship*, we observe rather unambiguous response patterns that allow good classification. This is more likely to occur when only a few positive ratings are obtained. For *Boat/Ship* all four raters agree on “positive” in two cases, and in six cases three raters vote “negative” against a single rater. Almost identically, for the concept *Bus* all four raters agree on

“positive” in one case, while three raters vote “negative” against a single rater in six cases. The latent class model can resolve the ambiguity very reliably and the estimated error is $E = 0$, despite the fact that there is some disagreement.

However, we observe generally a higher classification error for the more frequent concepts, in particular those of concept set b . We have already confirmed this effect in prior work (Volkmer et al. 2005), that is we expect a higher disagreement in tandem with more prevalent concepts. Indeed, based on our results, we compute a correlation coefficient of $r = 0.81$ between concept prevalence and classification error — a strong correlation.

We can compare the LCM-based classification error E to the proportion of classification errors based on the unconditional latent class probabilities $P(X = x)$. This results in the classification performance measure λ , that is defined as follows:

$$\lambda = 1 - \frac{E}{\max[P(X = x)]} \quad (4)$$

As the measure λ is independent of class prevalence, we believe it can directly be used to assess annotation quality. However, λ is not a measure for inter-rater agreement. It estimates how well the example images could be classified based on the observed ratings; it should be interpreted similar to an R^2 measure (Vermunt & Magidson 2003). A value of $\lambda = 1.0$ indicates perfect classification.

To support our view, we compare λ to the intraclass correlation index (Fleiss 1981). This correlation index is a variant of the well-known Kappa statistic for inter-rater agreement by Cohen (Cohen 1960) and is therefore often referred to as Fleiss’ Kappa. In contrast to Cohen’s Kappa, however, the intraclass correlation index can be used to assess more than two judgements.

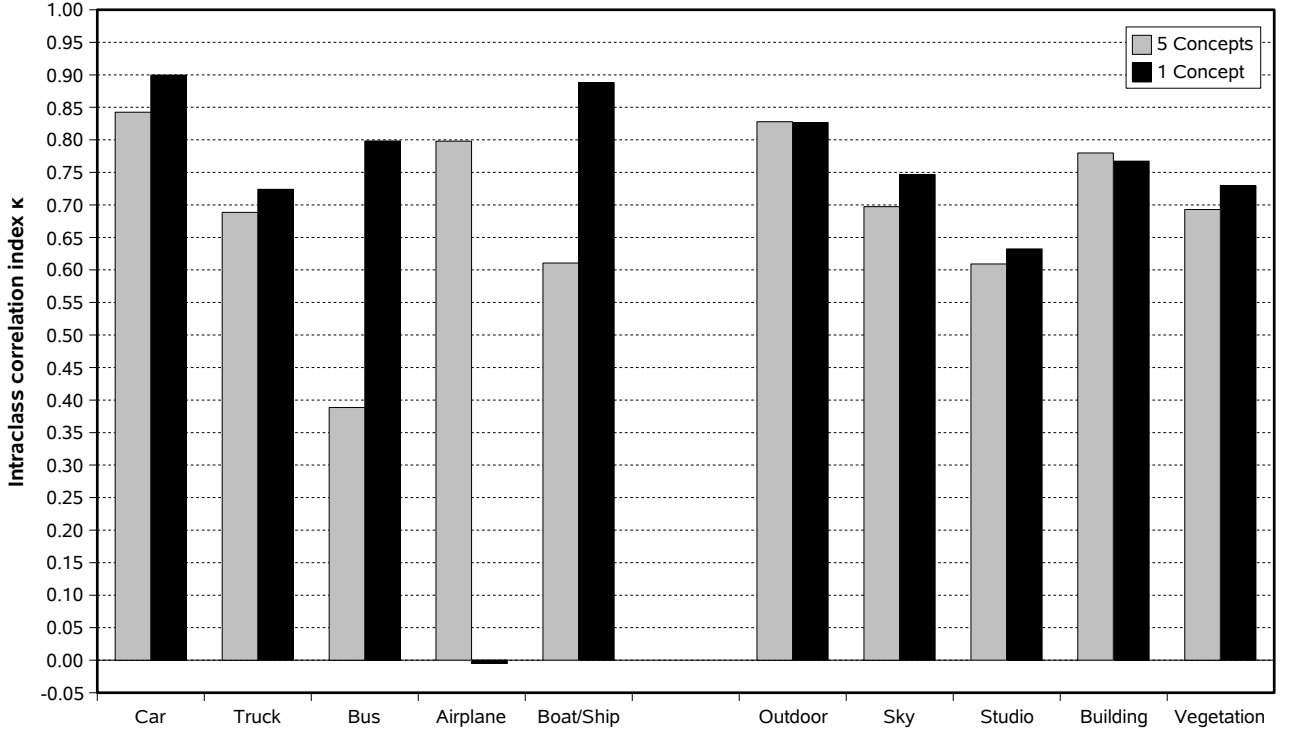


Figure 6: Intraclass correlation index κ for annotations done in single-concept mode compared to the intraclass correlation for annotations in multiple concept mode. We observe no statistically significant differences between both. For *Bus*, *Boat/Ship*, and *Airplane* in single-concept mode, the values for κ are most likely not reliable due to the very low prevalences of these concepts.

The intraclass correlation index is commonly denoted by κ and defined as follows:

$$\kappa = \frac{\bar{P} - \bar{P}_e}{1 - \bar{P}_e} \quad (5)$$

Where \bar{P} is:

$$\bar{P} = \frac{1}{N} \sum_{i=1}^N P_i \quad (6)$$

And \bar{P}_e is defined as:

$$\bar{P}_e = \sum_{j=1}^C p_j^2 \quad (7)$$

N is the number of samples, C is the number of latent classes. p_j is the proportion of all ratings to the j th of C classes. It is defined as follows:

$$p_j = \frac{1}{Nk} \sum_{i=1}^N n_{ij} \quad (8)$$

In Equation (8), n_{ij} is the number of judges who assigned the i th image to the j th class. The number of observed ratings per image is denoted by k , that is $k = 4$ in our experiment. The intraclass correlation index κ approaches 1.0 for perfect agreement. We compare κ and λ in Figure 5. The raw values for E , λ , and κ are shown in Table 3. Due to $E = 0$ for the concepts *Bus* and *Boat/Ship*, the LCM classification performance indicates perfect classification with $\lambda = 1.0$ in these cases. While there is some disagreement for these concepts, we believe that the low values of κ may not be reliable. As we pointed out above, the raw positive ratings do not imply as much ambiguity as κ suggests in these cases. For all other concepts, however, there is a strong correlation

between κ and λ . We compute the correlation factor for the eight most frequent concepts to be $r = 0.92$. Since we are not primarily interested in the level of rater agreement, but rather how well all images could be classified, we conclude that the LCM-based classification performance index λ is a good indicator for annotation quality when using multiple ratings.

Concept	E (LCM)	λ (LCM)	κ
Car	0.0024	0.9671	0.9113
Truck	0.0015	0.9174	0.8051
Bus	0.0000	1.0000	0.7244
Airplane	0.0014	0.8892	0.8101
Boat/Ship	0.0000	1.0000	0.8401
Outdoor	0.0173	0.9467	0.8684
Sky	0.0164	0.8935	0.8277
Studio	0.0385	0.8452	0.7698
Building	0.0125	0.9302	0.8554
Vegetation	0.0301	0.8068	0.7653

Table 3: Classification error E and classification performance λ based on latent class modelling compared to intraclass correlation index κ .

4.3 Inter-rater Agreement

We see from Figure 5 that the concepts *Bus*, *Studio* and *Vegetation* show the lowest agreement rates. However, the results for concept *Bus* in terms of κ may not be reliable because they are based on very few ratings. We therefore exclude it from the following analysis.

It appears that the concepts *Studio* and *Vegetation* were rather difficult to annotate for our participants. Indeed, 50% of all participants stated in the survey that they found it *difficult* or *very difficult* to annotate the concept *Studio*. On average, only 20% of all users perceived other concepts as difficult. Only 15% of all

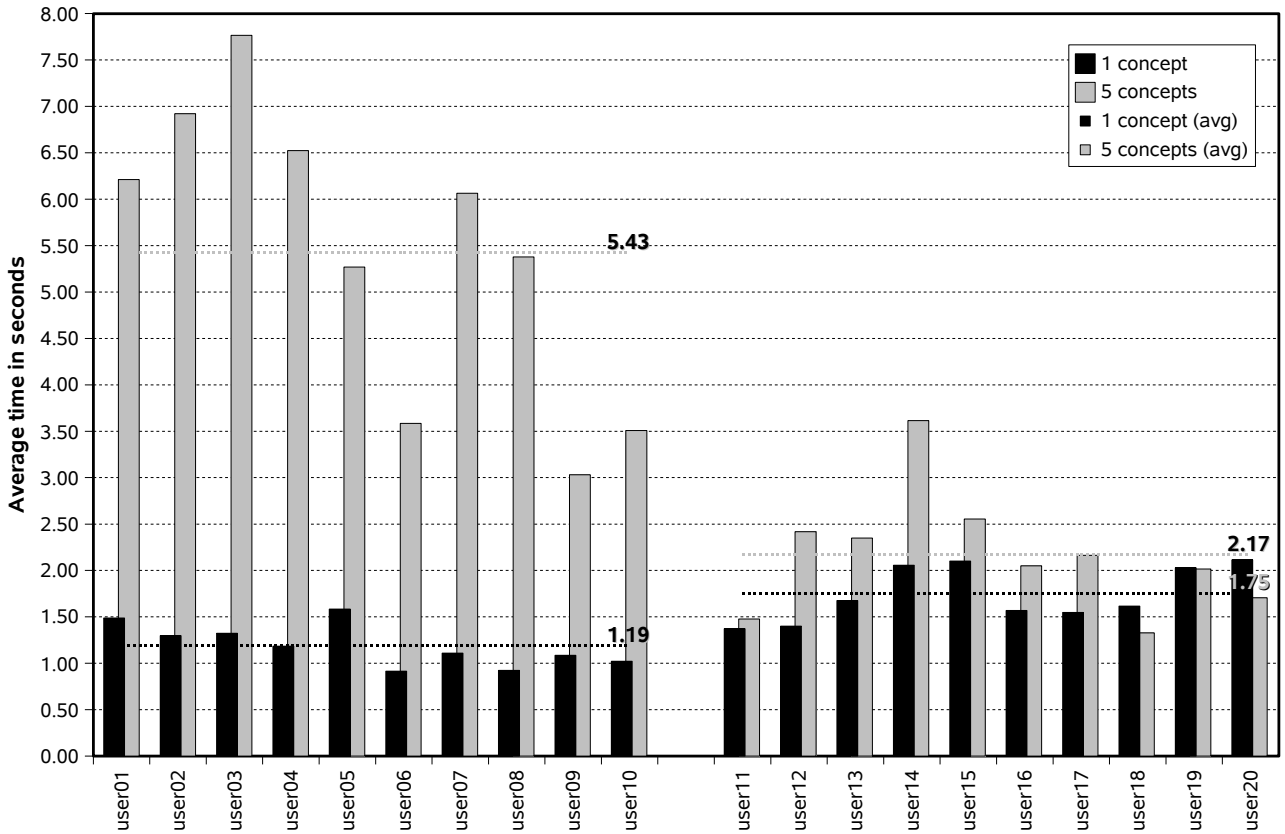


Figure 7: Annotation time per image by each user, naturally, only annotating one concept per image is faster. Interestingly, users 1 to 10 annotated their concepts in multiple-concept mode much slower than users 11 to 20. This is because users 1 to 10 annotated the more frequent concepts of group *b* in multiple-concept mode.

users rated *Vegetation* as difficult to annotate. However, this may only mean that users were not aware of its difficulty.

A potential weakness of our latent class modelling approach emerges when computing the model for very rare concepts, that is for sparse response tables (Vermunt & Magidson 2003). The LCM-based results for concept prevalence are highly likely to be reliable in such cases, but assessment of rater-agreement and annotation quality is difficult because the classification performance index λ may reach 1.0 despite notable disagreement. It is therefore important to understand λ as a purely statistical measure, that is as part of computing the model, no false classifications were estimated to have occurred. This does not guarantee, however, that no false classifications have happened in reality. Another problem of our model occurs when trying to apply it to fewer than three ratings per image. In this case, the system is overparameterised and we would need to introduce additional constraints. This does not seem possible in our case as it would mean, for example, that we have to assume one of our ratings to be absolutely reliable.

In the next step, we analyse the inter-rater agreement separated by annotation mode. For each image, we obtained four judgements, two that were done while the users annotated a single concept for each image, and two while multiple concepts were annotated. We use the intraclass correlation κ for this analysis. The results for κ in single-concept mode and multiple-concept mode are shown in Figure 6.

The computed value of $\kappa = -0.005$ for the concept *Airplane* in single-concept mode is not reliable because it is based on too few positive ratings. Only three positive ratings are observed, each of them in disagreement, which causes κ to drop to a value below zero. This may be interpreted as agreement be-

low chance, that is the agreement is lower than the agreement that can be expected by random assignment. Consistent with the problems that we previously observed when comparing κ and λ , we believe the κ measure is not necessarily reliable in cases with very few positive ratings. As can be seen from Figure 6, the differences for κ between single-concept mode and multiple-concept mode are most obvious for the three least prevalent concepts *Bus*, *Airplane*, and *Boat/Ship*. We will therefore treat these concepts as outliers in our analysis.

Regardless of some outliers, we observe differences between single-concept mode and multiple-concept mode. In most cases annotations done in single-concept mode have a higher intraclass correlation. While this seems to support our hypothesis that annotating several concepts at a time leads to higher disagreement, the differences we measure are not statistically significant.

Given that we are unable to reliably quantify the intraclass correlation for three of our concepts, and given that we cannot observe statistically significant differences for the other concepts, we cannot confirm this hypothesis. However, this conclusion is on the basis of using five concepts and we still believe that annotating too many concepts at a time may lead to higher disagreement. Further studies will be needed to determine a maximum number of simultaneous concepts that can be annotated while maintaining acceptable quality. Based on our current observations, we conclude that annotating five concepts at a time is well within the capacity of the average human annotator.

Similarly, we cannot confirm a dependency of the inter-rater agreement on the concept vocabulary based on our data. The average intraclass correlation for the concept sets *a* and *b* is nearly identical with

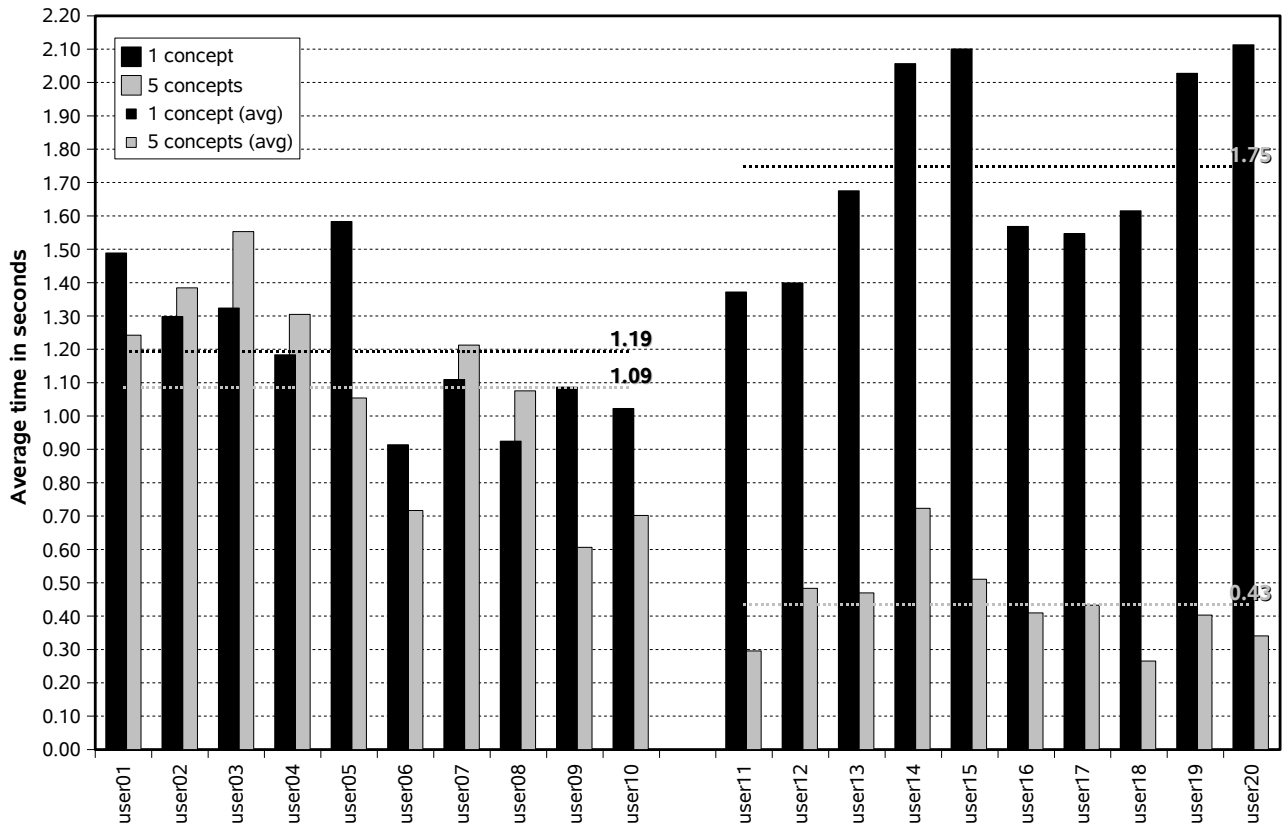


Figure 8: Annotation time calculated per image and per concept. We confirm that annotating the more frequently occurring concepts took on average longer than annotating the less frequent ones. Overall, annotating one image and one concept is done quickest when multiple concepts are annotated simultaneously.

$\kappa_a = 0.8173$ and $\kappa_b = 0.8182$. Unfortunately, the low frequencies of some concepts do not permit a more complete analysis. Further studies with a larger collection would be needed to draw more reliable conclusions.

4.4 Annotation Efficiency

We evaluated the timing data recorded during the annotation experiment. We measured the time that users spent while annotating each image. The average time that users spent per image is illustrated in Figure 7. We separated users 1 to 10 and users 11 to 20 into different groups and indicated their average times for both single and multiple-concept modes. From Figure 7 we can see that users 1 to 10 spent on average 5.43 seconds per image and five concepts while users 11 to 20 on average spent 2.17 seconds per image annotating five concepts. We believe the reasons for this effect is the fact that users 1 to 10 annotated the more prevalent concept group *b* in multiple concept mode. Users 11 to 20 annotated the concepts of group *a* simultaneously, these are significantly less frequent. For the same reason, users 1 to 10 were quicker in annotating in single-concept mode as they annotated the less prevalent concepts in this mode. Users 11 to 20 needed on average 1.75 seconds per image to annotate the more frequent concepts while users 1 to 10 needed on average 1.19 seconds for the less frequent concepts.

Our hypothesis is supported when evaluating the relative annotation times. We calculated the average annotation time needed per image and per concept in both modes and compared them in Figure 8. Again, we indicated the averages for users 1 to 10 and users 11 to 20 in both modes. We observe the shortest times for users 11 to 20 when they annotated the rare con-

cept set *a* in multiple-concept mode; only 0.48 seconds were on average needed per image and concept. Users 1 to 10 needed on average 1.09 seconds to annotate the more prevalent concepts of group *b* in combination. In conclusion, we can report that it is quicker to annotate in multiple-concept mode. Given that navigating between images in our annotation program causes an overhead, this is not surprising. Specialised implementations may address this by automatically guiding annotators to the next image after annotating one concept, but we do not believe that this will completely compensate the difference. Moreover, as we have confirmed that there is no significant increase in disagreement when annotating up to five concepts simultaneously, multiple concept annotation appears to be the preferable method to maximise efficiency. The average annotation time per concept and image among all participants in multiple-concept mode was 0.76 seconds. This is almost twice as fast as the 1.47 seconds that users on average needed to annotate in single-concept mode.

5 Conclusions and Future Work

We have conducted an experiment to study human judgement of digital imagery in regards to different annotation modes and pre-defined semantic concept categories. One goal of this study was to learn how to maximise efficiency while keeping disagreement between users to a minimum. Another goal was to explore the statistical combination of multiple user ratings for reliable image classification.

Overall, our results do not suggest any dependency of inter-rater agreement on different types of semantic concepts. However, individual concept specifications such as *Studio* or *Vegetation* need to be revisited as they seem to imply much ambiguity. Our initial hy-

pothesis, that annotating in multiple-concept mode leads to a decreased agreement between users, was not supported by our experiments. Annotating in multiple-concept mode may be the preferable strategy as long as the number of concepts does not exceed the capacity of the average annotator. Further experiments would be necessary to establish a threshold. In this light, and with regard to efficiency, it is preferable to annotate with multiple concepts simultaneously.

We have presented the application of latent class modelling for evaluating concept prevalences and classification performance. While the latter is not a measure of agreement, it strongly correlates with the inter-rater agreement and can serve as a quality index. We believe latent class analysis is a reliable statistic for this purpose and propose its application when training statistical learning algorithms. In particular, we propose the application of the LCM-based classification rules shown in Equations (1) and (2) to identify positive and negative examples. In the same way, LCM may be applied to combine results of multiple automatic classification algorithms during the retrieval phase. We believe that this can be helpful for implementing a combined retrieval approach. We will explore these applications of latent class modelling in greater depth in our future work.

As latent class models can be applied to varying numbers of raters (Uebersax & Grove 1990), we plan to revisit previously generated annotation data (Volkmer et al. 2005) for a new analysis.

We believe the outcome of this work forms the basis for better semantic indexing of image and video data.

Acknowledgements

We are very grateful to all 20 volunteers who have spent some of their valuable time to participate as annotators in our experiment.

References

- Cohen, J. (1960), 'A Coefficient of Agreement for Nominal Scales', *Educational and Psychological Measurement* **20**, 37–46.
- Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K. & Harshman, R. (1990), 'Indexing by Latent Semantic Analysis', *Journal of the American Society for Information Science* **41**(6), 391–407.
- Dumais, S. T. (1995), Latent Semantic Indexing (LSI): TREC-3 Report, in D. Harman, ed., 'NIST Special Publication 500-226: Proceedings of the Third Text REtrieval Conference (TREC 3)', Gaithersburg, MD, USA, pp. 219–230.
- Fleiss, J. L. (1981), *Statistical Methods for Rates and Proportions*, 2nd Ed., John Wiley & Sons, Inc., New York, NY, USA, chapter 13. The Measurement of Interrater Agreement, pp. 212–236.
- Hofmann, T. (1999), Probabilistic Latent Semantic Indexing, in D. Harman, ed., 'Proceedings of the ACM-SIGIR International Conference on Research and Development in Information Retrieval 1999', ACM Press, NY, USA, Berkeley, CA, USA, pp. 50–57.
- Lazarsfeld, P. F. & Henry, N. W. (1968), *Latent Structure Analysis*, Houghton Mifflin New York, New York, NY, USA.
- Lin, C.-Y., Tseng, B. L. & Smith, J. R. (2003), Video Collaborative Annotation Forum: Establishing Ground-Truth Labels on Large Multimedia Datasets, in E. M. Voorhees & L. P. Buckland, eds, 'TRECVID 2003 Workshop Notebook Papers', Gaithersburg, MD, USA.
<http://www.alpha-works.ibm.com/tech/videoannex>.
- Naphade, M., Kennedy, L., Kender, J., Chang, S.-F., Smith, J. R., Over, P. & Hauptmann, A. (2005), A Light Scale Concept Ontology for Multimedia Understanding for TRECVID 2005, Technical Report RC23612, IBM T.J. Watson Research Center, Hawthorne, NY, USA.
[http://domino.watson.ibm.com/library/CyberDig.nsf/papers/A33ABDB65967B5%3B852570070056B36F/\\$File/rc23612.pdf](http://domino.watson.ibm.com/library/CyberDig.nsf/papers/A33ABDB65967B5%3B852570070056B36F/$File/rc23612.pdf).
- Over, P., Ianeva, T., Kraaij, W. & Smeaton, A. F. (2005), TRECVID-2005 – An Introduction, in P. Over & T. Ianeva, eds, 'TRECVID 2005 Workshop Notebook Papers', Gaithersburg, MD, USA.
<http://www-nlpir.nist.gov/projects/tvpubs/tv5.papers/tv5intro.pdf>.
- Uebersax, J. S. (1992), 'A Review of Modeling Approaches for the Analysis of Observer Agreement', *Investigative Radiology* **27**(9), 738–743.
- Uebersax, J. S. & Grove, W. M. (1990), 'Latent Class Analysis of Diagnostic Agreement', *Statistics in Medicine* **9**, 559–572.
- Uebersax, J. S. & Grove, W. M. (1993), 'A Latent Trait Finite Mixture Model for the Analysis of Rating Agreement', *Biometrics* **49**, 823–835.
- Vermunt, J. K. (1996), Log-linear Models for Event Histories, PhD thesis, Department of Methodology and Statistics, Tilburg University, The Netherlands.
- Vermunt, J. K. (1997), LEM: A General Program for the Analysis of Categorical Data, Technical report, Department of Methodology and Statistics, Tilburg University, The Netherlands.
- Vermunt, J. K. (2003), Applications of Latent Class Analysis in Social Science Research, in T. D. Nielsen & N. L. Zhang, eds, '7th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU) 2003', Vol. 2711 of *Lecture Notes in Computer Science*, Lecture Notes in Computer Science, Aalborg, Denmark, pp. 22–36.
- Vermunt, J. K. & Magidson, J. (2003), 'Latent Class Analysis', *The Sage Encyclopedia of Social Sciences Research Methods* **3**, 549–553.
- Volkmer, T., Smith, J. R., Natsev, A., Campbell, M. & Naphade, M. R. (2005), A Web-based System for Collaborative Annotation of Large Image and Video Collections, in 'Proceedings of the ACM International Conference on Multimedia 2005', Singapore, pp. 892–901.
- von Ahn, L. & Dabbish, L. (2004), Labeling Images with a Computer Game, in 'Conference on Human Factors in Computing Systems (CHI) 2004', Vienna, Austria, pp. 319–326.
<http://www.espgame.org>.

Segmentation and Border Identification of Cells in Images of Peripheral Blood Smear Slides

Nicola Ritter

School of Information Technology
Murdoch University
Perth, W.A., Australia, 6150
n.ritter@murdoch.edu.au

James Cooper

Department of Computing
Curtin University of Technology
Perth, W.A., Australia, 6102
jc@cs.curtin.edu.au

Abstract

We present an unsupervised blood cell segmentation algorithm for images taken from peripheral blood smear slides. Unlike prior algorithms the method is fast; fully automated; finds all objects—cells, cell groups and cell fragments—that do not intersect the image border; identifies the points interior to each object; finds an accurate one pixel wide border for each object; separates objects that just touch; and has been shown to work with a wide selection of red blood cell morphologies. The full algorithm was tested on two sets of images. In the first set of 47 images, 97.3% of the 2962 image objects were correctly segmented. The second test set—51 images from a different source—contained 5417 objects for which the success rate was 99.0%. The time taken for processing a 2272x1704 image ranged from 4.86 to 11.02 seconds on a Pentium 4, 2.4 GHz machine, depending on the number of objects in the image.

Keywords: erythrocyte, segmentation, border detection, graph algorithm, blood cell.

1 Introduction

For both animals (Reagan, Sanders & DeNicola 1998) and humans (Bell 1997), slides of stained peripheral blood smears are examined to aid diagnosis. The slides include three types of cell: white blood cells (leukocytes), red blood cells (erythrocytes), and platelets. Analysis of these slides by technicians aims to identify pathological conditions that cause changes in the blood cells. For red blood cells (RBC) these changes include incursions (Reagan et al. 1998), as well as changes in cell shape, size and colour (Bessis 1973, Bessis 1977). There may also be erythrocyte fragments (shistocytes) (Lesesve, Salignac, Alla, Defente, Benbih, Bordigoni & Lecompte 2004), joined erythrocytes (rouleaux) (Reagan et al. 1998) or aggregated erythrocytes (agglutination) (Foresto, D'Arrigo, Carreras, Cuzzo, Valverde & Rasia 2000). The changes are linked to specific diseases and conditions (Bacus, Belanger, Aggarwal & Trobaugh Jr. 1976, Bell 1997),

This work was supported by a grant from the Division of Arts at Murdoch University. The slides and the first set of images were provided by A.Prof Phillip Clark of the Department of Clinical Pathology at Murdoch University. The second set of images were taken by Stanton Smith.

Copyright ©2007, Australian Computer Society, Inc. This paper appeared at The Thirtieth Australasian Computer Science Conference (ACSC2007), Victoria, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 62. Gillian Dobbie, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

making this an important diagnostic indicator. Slides are therefore examined to enable classification of the red blood cell morphology (Walton 1973).

The potential use of automated blood slide examination is large. Ingram and Preston Jr. (1970) estimated that in the US alone there would be over 1 million slide examinations every day. By 1976 the estimate had increased to over 50 million a day (Preston Jr. 1976). This number could be expected to have grown exponentially along with population since then.

However examination of slides by humans has a number of problems. It is time consuming (Preston Jr. 1976, Rowan 1986, Di Ruberto, Dempster, Khan & Jarra 2002) and therefore expensive (Ingram & Preston Jr. 1970). It is subjective (Connors & Wilson 1986, Payne, Bridgen & Edora 1996) and prone to human error (Bacus 1971, Fairbanks 1971), which means that it is difficult to get consistent results from the examination. Human analysis usually only results in a qualitative description of the presence of various morphologies and gives no quantitative results (Robinson, Benjamin, Cosgriff, Cox, Lapets, Rowley, Yatco & Wheelless 1994). This makes it difficult to track the progress of the condition and its treatment.

Currently there are several instruments that do partial differential counting of erythrocytes. A typical example of such an instrument is the DM96, produced by Cellavision (*CellaVision DM96 Technical specifications* 2006). However even this state-of-the-art instrument can only pre-characterise the red blood cells based on six general classifiers: polychromasia, hypochromasia, anisocytosis, microcytosis, macrocytosis, and poikilocytosis. More precise classification into the 30+ possible RBC morphologies (Bessis 1977, Reagan et al. 1998) is still done manually. Currently over 16% of automatically scanned slides are manually reviewed afterwards (Novis, Walsh, Wilkinson, Louis & Ben-Ezra 2006).

To be useful, a computer-based red blood cell differential count will need to be fully automatic (Rowan 1986), faster than a human observer, and at least as accurate as a human observer.

2 Literature Review

The history of research into automated blood slide examination dates back to 1975 (Bentley & Lewis 1975). However it is only recently that digital photography, computer speed, RAM size and secondary storage capacity have made the reality possible.

Analysis of blood slides must be fully automated to be useful (Rowan 1986). However the difficulty of the processing involved (Costin, Rotariu, Zbancioc, Costin & Hanganu 2001) has mostly limited papers to comparisons based on red cells segmented

either manually (Bentley & Lewis 1975, Albertini, Teodori, Piatti, Piacentini, Accorsi & Rocchi 2003) or semi-automatically (Bacus et al. 1976, Robinson et al. 1994, Dasgupta & Lahiri 2000, Costin et al. 2001, Gering & Atkinson 2004) from an image. Furthermore, in images from peripheral blood smear slides, the shading of the interior of the red blood cell as well as the overall shape and size of the cell (Reagan et al. 1998) exhibit meaningful variation. This necessitates identification of all points interior to the cell as well as accurate identification of the cell border. However no algorithm so far published finds both the border and the interior points of red blood cells, fully automatically.

Thresholding has been used to pre-process images as an aid to segmentation (Gonzalez & Woods 2002). However with red blood cell images this causes problems due to the pale nature of the interior of the cells, which then necessitates further processing. Adjouadi and Fernandez (2001) find the cell borders using eight-directional scanning within thresholded images of normal blood. However the method would not find the whole of the border of severely deformed cells such as those in the image of Fig. 1(b), as these contain edge points that would not be reached by any of the eight scan-lines. Moreover the process does not result in identification of the points within the contours.

Di Ruberto *et al.* (2000) follow thresholding with a segmentation method using morphological operators combined with the watershed algorithm. However their work is aimed at segmentation of red blood cells containing parasites and is designed to increase the compact nature and roundness of the cells. Such assumption of roundness is not appropriate for segmentation of RBC images for the purpose of classification, because these may contain deformed red blood cells as well as red blood cell fragments, for which the preservation of shape is important. Their method is also complicated—requiring 9 intermediate steps—and does not result in border identification.

Classical edge operators such as Sobel or Canny produce multiple thick edges as well as multiple edges interior to cells (Adjouadi & Fernandez 2001). Furthermore such an operator is merely a pre-processing technique which leaves the actual edge detection yet to be done. Some success has been found using graph theory (Martelli 1976, Pope, Parker, Clayton & Gustafson 1985, Fleagle, Johnson, Wilbricht, Skorton, Wilson, White, Marcus & Collins 1989, Fleagle, Thedens, Ehrhardt, Scholz & Skorton 1991) to navigate around edge pixels found in an edge image. However this work has involved images of single objects manually located in an image, and does not address the problems of multiple objects in the image; object location; removal of extraneous edges (internal to the cell); or the selection of suitable starting and ending points for the graph search.

Another approach to border detection is that of active contours, or snakes (Kass, Witkin & Terzopoulos 1988), which can be applied either to the original image or to an edge image. However when used to identify cell borders, the resulting contours do not correspond with the exact borders of the cells (Ongun, Halici, Leblebicioglu, Atalay, Beksac & Beksac 2001, Wang, He & Wee 2004), which would cause problems with later RBC classification, where the exact boundary shape is important. Other problems with the use of contours for images of peripheral blood smear slides include the initial positioning of the multiple contours required; the tendency of the contours to find the inner pale cell areas as well as or instead of the outer edges; and the failure of contours to identify pixels interior to the contour.

We present a fully automated algorithm that lo-

cates every object—cell or cell group—in an image from a peripheral blood smear slide. For each object it identifies both the pixels within the cell and a 1-pixel wide border of the cell.

3 Materials

As almost all mammalian red-cells are similar in shape, we use canine blood cell slides for our processing as canines have the largest cells of the non-human mammals (Reagan et al. 1998). Their blood slides are also more easily obtained than those of human blood.

The slides were made using the ‘wedge technique’. They were then air dried, fixed in alcohol and stained using Wright’s and Giemsa stain. The slides were mounted in a Nikon eclipse microscope and viewed at 100x magnification with oil immersion. Parts of the slide were then digitally photographed using a Nikon coolpix camera, to give 2272x1704 colour images. The colour images were converted to greyscale to improve initial processing time. Figure 1 shows typical greyscale images of canine peripheral blood smear slides. Figure 1(a) shows normal blood and Fig. 1(b) shows blood containing irregularly shaped, damaged red blood cells called acanthocytes. The large cell containing darker material—the nucleus—is a white blood cell, the smaller diffuse cells are platelets and the smaller solid pieces are red-cell fragments.

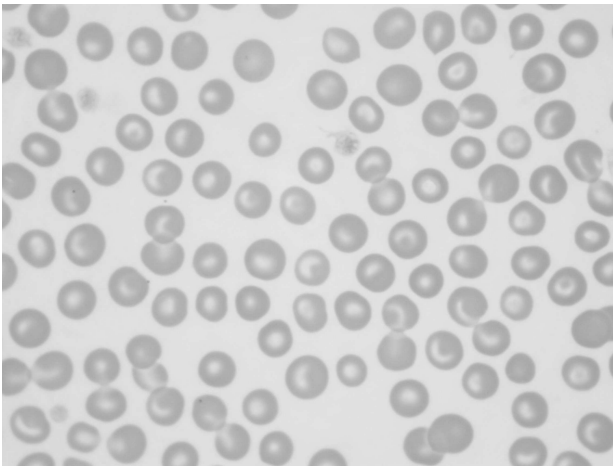
4 A Fully Automated Segmentation and Boundary Identification Algorithm

4.1 Segmentation

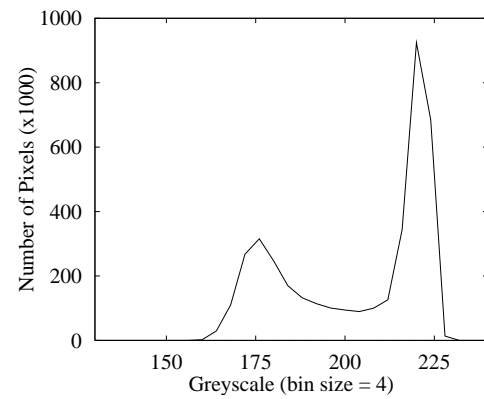
Greyscale histograms of the images in Fig. 1 are shown in Fig. 2. They were calculated using a bin size of 4, which acts to smooth the histogram. As can be seen, there is a marked similarity between the histograms. This similarity holds for all images to which we have access, suggesting that the histograms can be used for automatic selection of a useful threshold (Weszaka, Nagel & Rosenfeld 1974). This is done as follows. A search is made from the right to the left of the histogram, for the first decrease in pixel count. This gives the location of the right peak, which corresponds to the most common greyscale within the background. From there the search continues until the first increase in pixel count, which gives the middle low point. As this low point includes pixels that form part of the border of the cells, the threshold is chosen as the greyscale that falls $\frac{1}{4}$ of the ‘distance’ from the central minimum towards the right peak. This choice forms a good balance between separation of overlapping cells and ‘leakage’ of the background into central pale areas close to the border of a cell.

The background of the image is then identified using a 4-adjacency connected components algorithm (Gonzalez & Woods 2002) instead of thresholding. The initial seed point of this algorithm is the top-left most point in the image with a greyscale greater than the calculated threshold. From this seed point all connected pixels with greyscale greater than the threshold value are converted to white. To ensure that the seed point is not interior to a cell, the connected pixels are counted. If this number is much less than the number of pixels between the central minimum of the histogram and the right most peak, then the algorithm iterates using a new seed point, 10 pixels down and to the right of the current seed point.

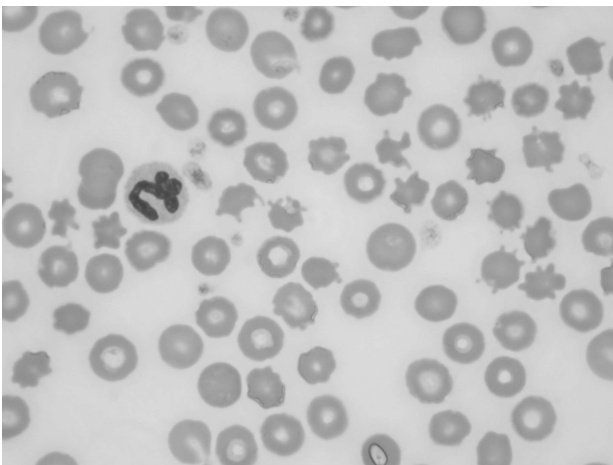
The algorithm results in a segmented image with the background white and the foreground containing all the cells (red, white and platelets). Incomplete cells that overlap the edge of the image are deleted



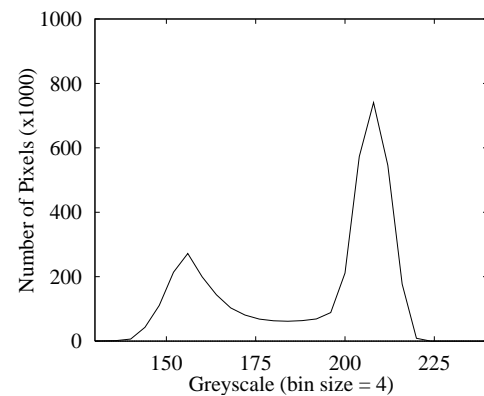
(a) Normal blood.



(a) The histogram of the image shown in Fig. 1(a).



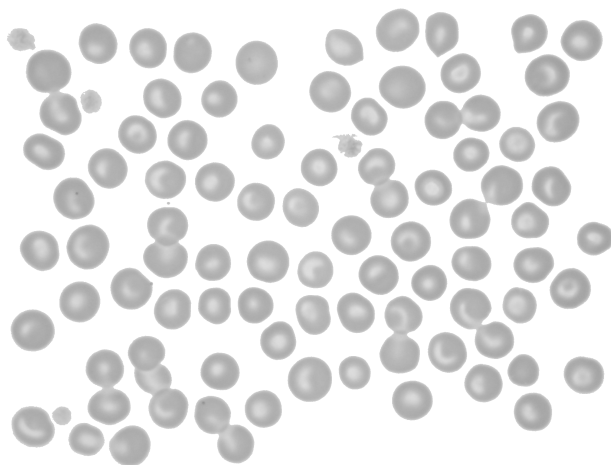
(b) Abnormal blood.



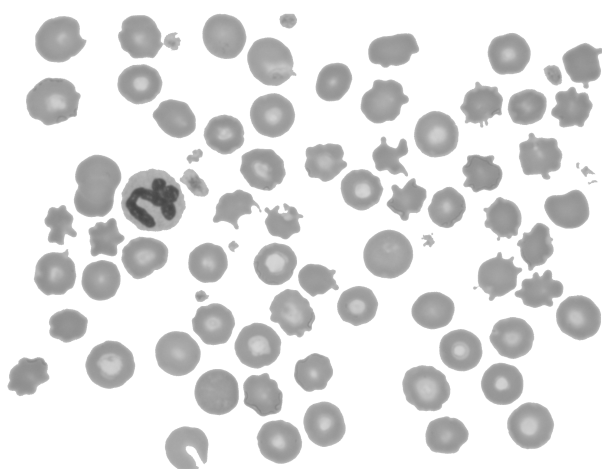
(b) The histogram of the image shown in Fig. 1(b).

Figure 1: Images taken from slides of peripheral blood smears of canine blood.

Figure 2: Histograms for images taken from peripheral blood smear slides show marked similarities to each other.



(a) Processed from Fig. 1(a).



(b) Processed from Fig. 1(b)

Figure 3: Images in which the objects have been separated from the background, and those that overlap the boundary of the image have been removed.

by doing a ‘circuit’ of the image removing all pixels that are connected to a non-white boundary point. Figure 3 shows the resulting image for the two original images of Fig. 1.

The objects in the image are then identified as separate entities as follows. A copy is made of the image. A search is then made from the top-left most corner of the copy for the first non-white pixel. This pixel is used as a seed point for a 4-adjacency connected components algorithm that gathers all connected pixels that are not white. Each pixel so gathered is designated either a border pixel—if 8-connected to a white pixel—or an interior pixel, and added to the new object. The choice of 8-connected rather than 4-connected border pixel selection is made due to the improved smoothness of the final borders after the processing described in the next section. All the pixels gathered into the object—both border and internal—are then set to white and the search repeated.

The result is an array of all objects within the original image, where the objects are cells or cell groups. Each object is itself composed of two arrays: points interior to the cell, and points on the border of the object.

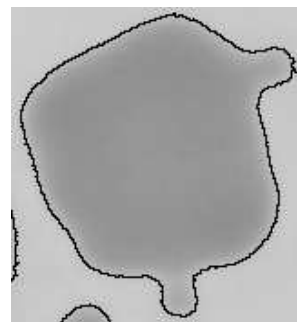


Figure 4: A cell from the top left of the image in Fig. 1(a). The black line shows the initial border formed using the algorithm described in Section 4.1.

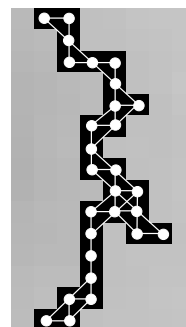


Figure 5: A portion of the border of the cell shown in Fig. 4: it contains ‘stubs’ that should not be part of the final border. Superimposed on the image is the graph that models the border, with pixels becoming vertices. Arcs are formed from 8-connectedness of the border pixels.

4.2 Border Thinning

The result of the segmentation and initial border selection can be seen in the closeup image in Fig. 4, where the border is shown in black. However the borders are wide and have many small extra stubs and blocks that would interfere with subsequent shape analysis. In particular statistical analysis can be biased by borders that have clusters of pixels. It is therefore useful to refine the borders by pruning stubs and thinning the border to one pixel width.

To refine a border, it is defined as representing a sparse graph, where the vertices are pixel positions. Arcs in the graph are formed between any pair of vertices representing 8-connected pixels. This is illustrated in Fig. 5, which shows a portion of the right-hand side of the border of the cell shown in Fig. 4.

4.2.1 Graph Representation

The traditional way to represent a graph is via an adjacency list or adjacency matrix (Sedgwick 2001). However building such a representation from the array of border points is slow as every point must be compared to every other point to check for connectivity. We therefore use an alternate representation. The border points are sorted—using a row-order index sort—into an array of arrays. In this two dimensional structure, each row stores the indices of border points that fall in the same image row. This allows direct access to all border points in a specific row in the image using the image row value itself—minus the value of the lowest row—as an array index. Each row of indices is also sorted on column order, further improving search time. A schematic diagram of part of the resulting structure is shown in Fig. 6.


```

Use the initial vertex as a one vertex SPT
WHILE less than V-1 vertices added AND
    target not found
    Search the SPT for the shortest path
    between the starting vertex and a
    vertex adjacent to the SPT
    Add that vertex to the SPT
    IF the new vertex is the target
        found = true;
    ENDIF
ENDWHILE

```

Algorithm 1: Dijkstra's shortest path algorithm (Sedgwick 2001), modified to find the shortest path between two specific vertices.

Whilst this representation could be used to build an adjacency list or matrix, this proves to be unnecessary as the search for a connected vertex is already reduced to a search of only the three short arrays containing indices of points above, below and in the same image row as the current vertex. Its use resulted in a 50% reduction in the time taken to refine the border as compared with building an adjacency matrix or list from the original border array.

4.2.2 Graph Algorithm

Dijkstra's shortest path algorithm (Sedgwick 2001) builds a shortest path tree (SPT) from a starting vertex to all other vertices. This is achieved by iteratively searching for the shortest total path from the starting vertex and a vertex adjacent to one already in the SPT, and then adding the new vertex to the SPT. This algorithm can be used to find the shortest path between two specific vertices by prematurely ending the SPT build when the target vertex is found. Algorithm 1 contains pseudo-code for this.

However, searching "the SPT for the shortest path between the starting vertex and a vertex adjacent to the SPT" is time consuming, as it involves iterating through every node in the current tree. However, for our graph this can be simplified. Since the graph only contains previously identified potential border points, at each iteration of the algorithm only an arc of either length 1 or $\sqrt{2}$ can be added. This is because the cost function is based purely on physical distance and does not include image information as for previous users of graph theory (Martelli 1976, Pope et al. 1985, Fleagle et al. 1991). Furthermore, since all distances from the starting vertex are given by $m + n\sqrt{2}$, the tree can be formed using discrete layers sorted on distance from the root. Therefore when building the SPT, only the highest unprocessed layer needs to be processed: it is guaranteed that the next two layers that should be added will be the ones that are 1 unit and $\sqrt{2}$ units longer than that layer. A schematic diagram of the start of the tree-build is shown in Fig. 7 and the simplified algorithm is shown in Algorithm 2. It results in an average time-saving of 73% as compared with using Dijkstra's original algorithm.

4.2.3 Choice of End Points

The final problem to be solved is the choice of the starting and target pixels. Since the purpose is to find the shortest path around the previously selected border pixels, the obvious choice would be two adjacent pixels. However this has several problems. Firstly the SPT would find the path between the two that was just 1 or $\sqrt{2}$ long and secondly any particular pair of pixels chosen might be on a spur that should be removed. We solve these problems by finding a set of

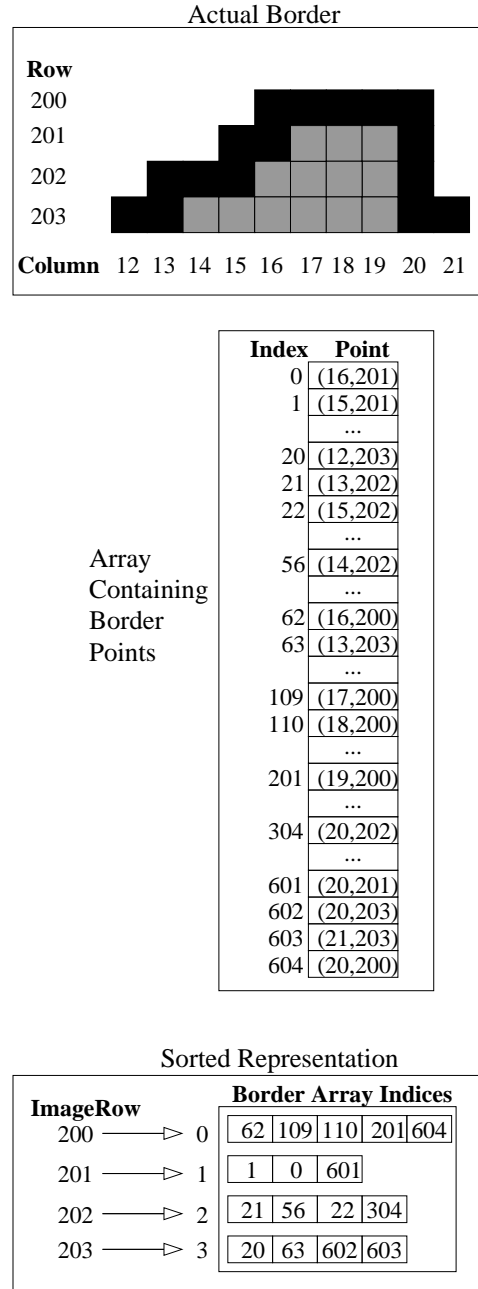


Figure 6: A schematic diagram showing the sorted representation of the original border points. This representation allows the search for two adjacent border points to be reduced to a search of three short arrays.

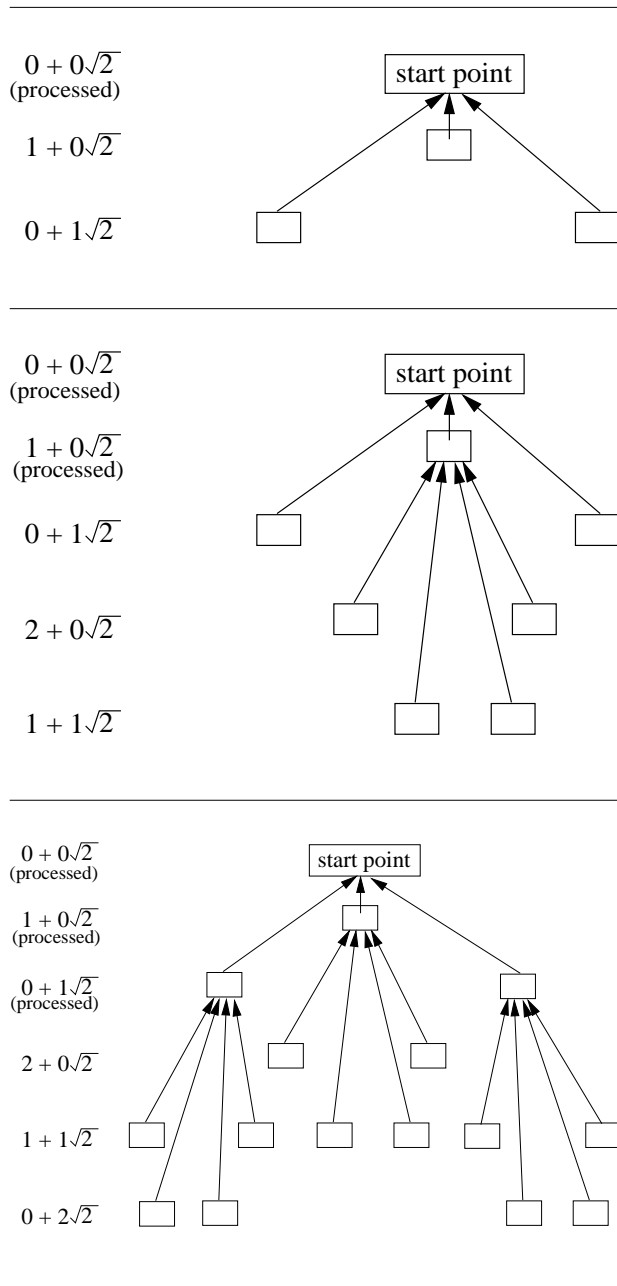


Figure 7: A schematic diagram showing how the layers of the shortest path tree are built using Algorithm 2.

```

Use the initial vertex as a one vertex SPT
WHILE unprocessed layers in the SPT AND
    target not found
    layer = next unprocessed layer in the SPT
    FOR each node in the layer AND
        while target not found
            Add to layer1 all border nodes
            4-adjacent to this node, that are
            not already in the SPT
        ENDFOR
        Add layer1 to correct place in the SPT
        FOR each node in the layer AND
            while target not found
                Add to layer2 all border nodes
                4-diagonal to this node, that are
                not already in the SPT
            ENDFOR
        Add layer2 to correct place in the SPT
        Mark layer as processed
    ENDWHILE IF target not found
        return error
    ELSE return noerror
ENDIF

```

Algorithm 2: A modified version of Dijkstra's SPT algorithm. A schematic diagram showing the building of layers in the SPT, can be seen in Fig. 7.

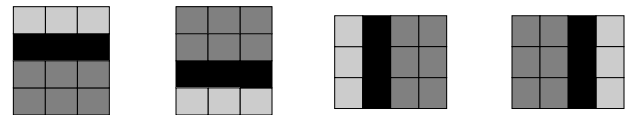


Figure 8: The border point configurations used to find good start and target border points for the SPT. The black pixels represent border pixels, the dark grey pixels are those that are part of the object and the light grey pixels represent the image background.

three points that form a straight line border at a place in the object that is at least two pixels wide. This results in a search for a set of three border points that conform to one of the four configurations shown in Fig. 8. The middle pixel of the three is then marked as already added to the tree and the other two used as the start and target points of Algorithm 2. Once the shortest path between them has been found, the middle point is added onto the end of the path. The result is an ordered one pixel wide border for the object. The points that were in the original border but not in the final border are added to either the interior of the object or the image background, as appropriate. An example of the final border is shown in Fig. 9.

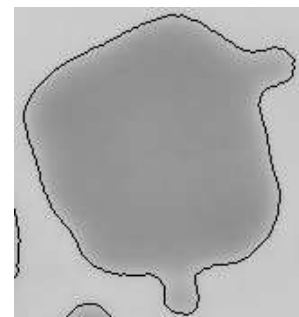
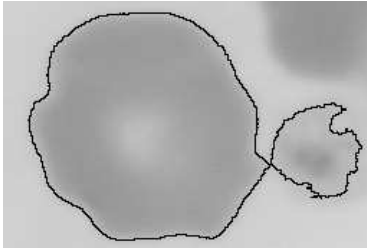
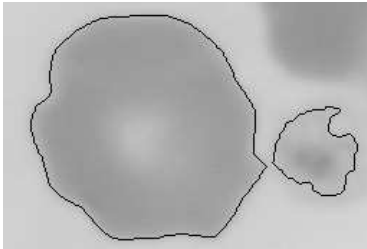


Figure 9: The same cell as that shown in Fig. 4 after the border has been refined.



(a) An object composed of two touching objects: the border forms a figure of 8.



(b) The two objects separated by eroding the original object, and reprocessing the two separate ones.

Figure 10: Two just-touching objects shown before and after the separation algorithm has been performed.

4.2.4 Correcting Figure-of-Eight SPT Failure

Occasionally two objects are just touching which results in a border with a figure-of-eight shape. An example of this can be seen in Fig. 10(a). Clearly the shortest path algorithm will fail in this case as the node at the intersection of the two border loops needs to be included twice, but can only be selected once. In this situation the algorithm returns an error code. Upon receiving this error code the object will be isolated from the rest of the image, eroded by removing the original border and reprocessed to produce two separate objects with smoothed borders. The result of this process can be seen in Fig. 10(b).

5 Results

The end result of the entire process described in Section 4 is an array of objects with both internal and border points identified. Figure 11 shows examples of the results, with the borders shown in black.

The method has been tested on 47 images of both normal (14) and diseased (33) canine blood. The objects in the images include normal red blood cells (discoytes), deformed red blood cells (echinocytes, acanthocytes, codocytes, pre-keratocytes, polychromatophils and rouleaux), red blood cell fragments (shistocytes), platelets, white blood cells and overlapping cells. The images are all 2272x1704 pixels in size. In these images there are 2962 objects that do not touch the boundary of the image. Our segmentation algorithm finds 2961 (99.97%) of them. The one missed is a pale platelet. Of the 2961 objects found, 97.33% are correctly segmented and bordered using our algorithm. The source of the errors can be seen in Table 1. The average processing time per image is 6.045 seconds, or 0.095 seconds per object, using a Pentium 4, 2.4 GHz machine with 1 GB of RAM.

Description	Number of Objects	% of Total
Outer region of platelet not included	41	1.36%
Platelet segmented as more than one object	12	0.41%
Pale outer tips of cell protruberances not included in cell segmentation	11	0.37%
Pale outer tips of cell protruberances segmented as separate objects	5	0.17%
'Blisters' on pre-keratocytes not included in cell segmentation	5	0.17%
Part of blister on pre-keratocytes segmented as separate object	1	0.03%
'Leakage' of the background into the pale inner area of the cell	4	0.16%
Correctly segmented with correct border identification	2882	97.33%
TOTAL	2961	100.00%

Table 1: Results of the segmentation and border identification algorithm when run on 47 images which include normal red blood cells, deformed red blood cells, red blood cell fragments, white blood cells, and platelets.

The algorithms were then run on 51 new images—17 normal, 34 diseased—taken from different slides by a different photographer but with similar equipment. These images are also all 2272x1704 pixels in size. The camera settings were varied to get different contrast, brightness and magnification. In this set of images there are 5417 objects of which our algorithm again misses one (a platelet). The success rate of correct segmentation and border identification of the objects found is 98.98%: a similar rate as with the first set of images. Information about the failures is given in Table 2. For this set of images the average processing time per image is 8.00 seconds, or 0.075 seconds per object. The time taken per image is greater due to the larger numbers of objects in the images, however the processing time per object is less due to the smaller pixel size of the average object.

6 Discussion

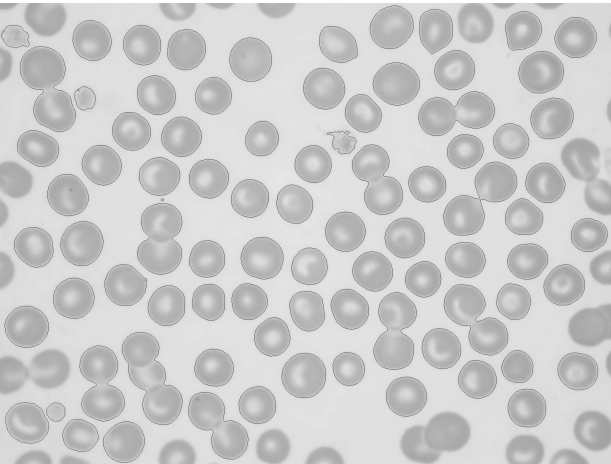
The number of fails for the segmentation and border detection of objects in the image is very small, especially when compared to the very high rates of observer error for human detection (Fairbanks 1971). Furthermore—as described below—most of the errors are not significant or can be dealt with in later processing.

Over two thirds of those objects not correctly segmented are platelets, which have diffuse areas that are not picked up by the segmentation algorithm. As the main work is aimed at differentiating red blood cells which means that platelets will be discarded, this is not a major problem. The automatic identification of platelets will be dealt with in future work.

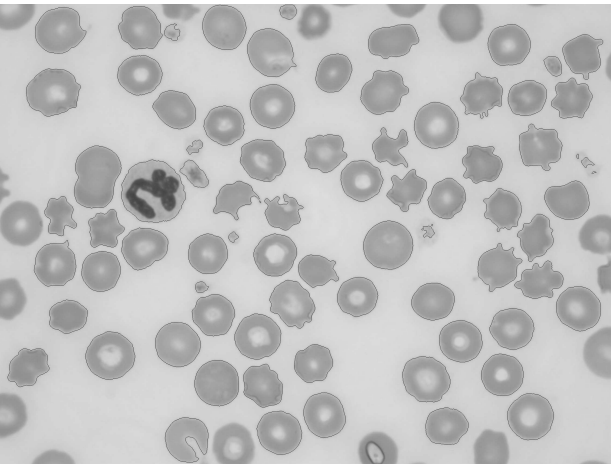
The loss of pale tips of protuberances is also not considered a major problem as in all but one case the cell has multiple other protuberances which would clearly define it as being deformed. The loss of the outer edge of a 'blister' in a pre-keratocyte cell, as shown in Fig. 12, would cause the cell to be later classified as a keratocyte, rather than a pre-keratocyte. Again this does not constitute a major problem, as later algorithms could check all cells classified as keratocytes for pale borders remaining around the blister. Furthermore, both morphologies are indicative of the same diseases (Reagan et al. 1998).

Description	Number of Objects	Percentage of Total
Outer region of platelet not included	39	0.72%
Platelet segmented as more than one object	3	0.06%
Pale outer tips of cell protruberances not included in cell segmentation	1	0.02%
Pale outer tips of cell protruberances segmented as separate objects	0	0.00%
'Blisters' on pre-keratocytes not included in cell segmentation	10	0.18%
Part of blister on pre-keratocytes segmented as separate object	1	0.02%
'Leakage' of the background into the pale inner area of the cell	1	0.02%
Correctly segmented with correct border identification	5361	98.98%
TOTAL	5416	100.00%

Table 2: Results of the segmentation and border identification algorithm when run on 51 new images which again include normal red blood cells, deformed red blood cells, red blood cell fragments, white blood cells, and platelets.



(a) The final borders found in the image of Fig. 1(a).



(b) The final borders found in the image of Fig. 1(b).

Figure 11: Two images from peripheral blood smear slides showing the final borders in black. All pixels inside the borders have been collected as part of the object.

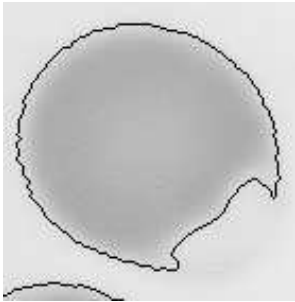


Figure 12: A pre-keratocyte cell where the outer pale edge of the blister has not been included by the segmentation algorithm. The cell therefore looks like a keratocyte instead.

Finally it is worth noting that it is important that the algorithm does not separate overlapping cells. This is because overlapping cells may in fact be joined cells (rouleaux or aggregates) which are caused by disease. It is therefore necessary that cell groups be left intact for later classification.

7 Conclusion

We present a fully automatic method for segmentation and border identification of all objects that do not overlap the boundary in an image taken from a peripheral blood smear slide. Unlike prior algorithms the method is fully automated, fast and accurate. It can separate cells that just touch and has been shown to work with both normal, deformed and joined red blood cells, as well as white blood cells and red cell fragments. It has been tested on a total of 98 images from two different sources with a high success rate.

The algorithm combines automatic threshold selection with connected-components and a novel adaptation of Dijkstra's shortest path algorithm and an alternate graph representation to the standard adjacency list or matrix. The result is an average processing time of 7.06 seconds per image, with an average of 84 objects per image.

Further work will involve use of shape factors and interior greyscale analysis to classify all of the red blood cells. This will enable a full red blood cell morphology count.

References

Adjouadi, M. & Fernandez, N. (2001), 'An orientation-independent imaging technique for the classification of blood cells', *Particle & Particle Systems Characterization* **18**(2), 91–98.

Albertini, M., Teodori, L., Piatti, E., Piacentini, M., Accorsi, A. & Rocchi, M. (2003), 'Automated analysis of morphometric parameters for accurate definition of erythrocyte cell shape', *Cytometry Part A* **52A**(1), 12–18.

Bacus, J. (1971), 'The observer error in peripheral blood cell classification', *American Journal of Clinical Pathology* **59**, 223–230.

Bacus, J., Belanger, M., Aggarwal, R. & Trobaugh Jr., F. (1976), 'Image processing for automated erythrocyte classification', *Journal of Histochemistry & Cytochemistry* **24**(1), 195–201.

Bell, A. (1997), Morphological evaluation of erythrocytes, in E. A. Stiene-Martin, C. A. Lotspeich-Steininger & J. A. Koepke, eds, 'Clinical Hema-

- tology: Principles, Procedures, Correlations', 2nd edn, Lippincott Williams & Wilkins, chapter 8, pp. 87–105.
- Bentley, S. & Lewis, S. (1975), 'The use of an image analysing computer for the quantification of red cell morphological characteristics', *British Journal of Haematology* **29**, 81–88.
- Bessis, M. (1973), Red cell shape: An illustrated classification and its rationale, in M. Bessis, R. Weed & P. Leblond, eds, 'Red Cell Shape: Physiology, pathology, ultrastructure', Springer-Verlag, New York, pp. 1–26.
- Bessis, M. (1977), *Blood smears reinterpreted*, Springer International, Berlin. Translated by G. Brecher.
- CellaVision DM96 Technical specifications (2006). <http://www.cellavision.com/>.
- Connors, D. & Wilson, M. (1986), 'A new approach to the reporting of red cell morphology', *Journal of Medical Technology* **3**(2), 94–785.
- Costin, H., Rotariu, C., Zbancioc, M., Costin, M. & Hanganu, E. (2001), 'Fuzzy rule-aided decision support for blood cell recognition', *Journal of Fuzzy Systems & Artificial Intelligence* **7**(1–3), 61–70.
- Dasgupta, A. & Lahiri, P. (2000), 'Digital indicators for red cell disorder', *Current Science* **78**(10), 1250–1255.
- Di Ruberto, C., Dempster, A., Khan, S. & Jarra, B. (2000), Segmentation of blood images using morphological operators, in 'Proceedings of the 15th International Conference on Pattern Recognition (ICPR 2000)', IEEE, Barcelona, Spain, pp. 397–400.
- Di Ruberto, C., Dempster, A., Khan, S. & Jarra, B. (2002), 'Analysis of infected blood cell images using morphological operators', *Image And Vision Computing* **20**(2), 133–146.
- Fairbanks, V. F. (1971), 'Is the peripheral blood film reliable for the diagnosis of iron deficiency anaemia?', *American Journal of Clinical Pathology* **55**, 447–451.
- Fleagle, S., Johnson, M., Wilbricht, C., Skorton, D., Wilson, R., White, C., Marcus, M. & Collins, S. (1989), 'Automated analysis of coronary arterial morphology in cineangiograms: geometric and physiologic validation in humans', *IEEE Transactions on Medical Imaging* **8**(4), 387–400.
- Fleagle, S., Thedens, D., Ehrhardt, J., Scholz, T. & Skorton, D. (1991), 'Automated identification of left ventricular borders from spin-echo magnetic resonance images', *Investigative Radiology* **26**(4), 295–303.
- Foresto, P., D'Arrigo, M., Carreras, L., Cuezco, R., Valverde, J. & Rasia, R. (2000), 'Evaluation of red blood cell aggregation in diabetes by computerized image analysis', *Medicina-Buenos Aires* **60**(5), 570–572.
- Gering, E. & Atkinson, C. (2004), 'A rapid method for counting nucleated erythrocytes on stained blood smears by digital image analysis', *Journal of Parasitology* **90**(4), 879–881.
- Gonzalez, R. & Woods, R. (2002), *Digital Image Processing*, Prentice Hall, New Jersey.
- Ingram, M. & Preston Jr., K. (1970), 'Automatic analysis of blood cells', *Scientific American* **223**(5), 72–82.
- Kass, M., Witkin, A. & Terzopoulos, D. (1988), 'Snakes: Active contour models', *International Journal of Computer Vision* **4**, 321–331.
- Lesesve, J., Salignac, S., Alla, F., Defente, M., Benbih, M., Bordigoni, P. & Lecompte, T. (2004), 'Comparative evaluation of schistocyte counting by an automated method and by microscopic determination', *American Journal of Clinical Pathology* **121**(5), 739–745.
- Martelli, A. (1976), 'An application of heuristic search methods to edge and contour detection', *Communications of the ACM* **19**(2), 73–83.
- Novis, D., Walsh, M., Wilkinson, D., Louis, M. S. & Ben-Ezra, J. (2006), 'Laboratory productivity and the rate of manual peripheral blood smear review - a College of American Pathologists q-probes study of 95141 complete blood count determinations performed in 263 institutions', *Archives of Pathology & Laboratory Medicine* **130**(5), 596–601.
- Ongun, G., Halici, U., Leblebicioglu, K., Atalay, V., Beksac, S. & Beksac, M. (2001), 'Automated contour detection in blood cell images by an efficient snake algorithm', *Nonlinear Analysis-Theory Methods & Applications* **47**(9), 5839–5847.
- Payne, N., Bridgen, M. & Edora, F. (1996), 'A redesign of RBC morphology reporting', *Medical Laboratory Observer* **28**(4), 60–65.
- Pope, D., Parker, D., Clayton, P. & Gustafson, D. (1985), 'Left ventricular border recognition using a dynamic search algorithm', *Radiology* **155**(2), 513–518.
- Preston Jr., K. (1976), Clinical use of automated microscopes for cell analysis, in K. Preston Jr. & M. Onoe, eds, 'Digital Processing of Biomedical Images', Lenum Press, New York, pp. 47–58.
- Reagan, W., Sanders, T. & DeNicola, D. (1998), *Veterinary hematology: atlas of common domestic species*, Iowa State University Press.
- Robinson, R., Benjamin, L., Cosgriff, J., Cox, C., Lapets, O., Rowley, P., Yatco, E. & Wheelless, L. (1994), 'Textural differences between AA and SS blood specimens as detected by image-analysis', *Cytometry* **17**(2), 167–172.
- Rowan, R. (1986), Automated examination of the peripheral blood smear, in 'Automation and quality assurance in haematology', Blackwell Scientific, Oxford, chapter 5, pp. 129–177.
- Sedgwick, R. (2001), *Algorithms in C++ Part 5: Graph Algorithms*, 3rd edn, Addison-Wesley.
- Walton, J. (1973), 'Uniform grading of hematologic abnormalities', *American Journal of Medical Technology* **39**(12), 517–523.
- Wang, X., He, L. & Wee, W. (2004), 'Deformable contour method: A constrained optimization approach', *International Journal of Computer Vision* **59**(1), 87–108.
- Weszaka, J. S., Nagel, R. N. & Rosenfeld, A. (1974), 'A threshold selection technique', *IEEE Transactions on Computing* **23**, 1322–1326.

Cross-Layer Verification of Type Flaw Attacks on Security Protocols

Benjamin W. Long¹Colin J. Fidge²David A. Carrington¹

¹School of Information Technology and Electrical Engineering
The University of Queensland, Brisbane, Australia

²School of Software Engineering and Data Communications
Queensland University of Technology, Brisbane, Australia

Abstract

Security protocols are often specified at the *application layer*; however, application layer specifications give little detail regarding message data structures at the *presentation layer* upon which some implementation-dependent attacks rely. In this paper we present an approach to verifying security protocols in which both the application and presentation layers are modelled. Using the Group Domain of Interpretation protocol as an example, our application layer specification of the protocol is used as input to the AVISPA model checking tool for analysis. Two type flaw attacks are found via model checking which are then verified against the corresponding presentation layer specification, thus identifying the minimal requirements to prevent the attacks.

1 Introduction

Electronic communication is now the predominant means of interaction for commercial, industrial, and private use. In response to this trend, it is important to ensure that transmitted information is not compromised by malicious parties, especially in areas such as defence, medicine, and commerce, where information leakage and corruption could have catastrophic consequences. In order to shield ourselves against potential threats, communication messages are secured by application of cryptographic functions and sent as part of ordered message sequences called *security protocols*.

Informal narrations that mix natural language and ad hoc notations (Abadi 2000), such as the *standard notation* (Carlsen 1994), conveniently describe security protocols at the application layer — the level at which message content is determined. However, standard notation descriptions do not indicate precisely what internal actions are required by protocol agents implementing them, nor does it suggest desirable properties of message items or cryptographic functions used. Furthermore, the standard notation gives little detail regarding message data structures at the presentation layer — the level at which the low-level representation of messages is determined — upon which some implementation-dependent attacks such as *type flaw attacks* rely. A lack of precise details at either of these layers can lead to misunderstandings and disputes over protocol correctness (Boyd 1990, Boyd & Mao 1993).

Formal methods provide well-defined languages that allow precise specifications to be written and

subsequent rigorous verification procedures to be performed. Hence, the use of formal methods is advocated for the design and analysis of security protocols (Gollmann 2003). The *Common Criteria* (The Common Criteria Project Sponsoring Organisations 1999), an internationally recognised set of criteria for evaluating security critical products, demands strict application of formal methods to the development process for achieving the highest assurance levels. Therefore, we are interested in the use of formal methods for specifying precise details of security protocols, and for analysing specifications of security protocols for potential attacks.

Type flaw attacks (Boyd 1990) occur at the presentation layer when intruders send unexpected messages to protocol agents who subsequently misinterpret the bit string encoding of message item types. For example, a type flaw attack may occur when a single message item of one type is confused with a single item of another type, or when a single message item of one type is confused with the concatenation of two or more other items of varying types. Type flaw attacks can be prevented by the use of ‘tags’ (Heather, Lowe & Schneider 2000), but problems can arise when a protocol interacts with another protocol that does not use a tagging scheme, or tags data in a different way (Meadows, Syverson & Cervesato 2004) (such as the attack on the GDOI protocol described in Section 6).

Nevertheless, we seek to find the necessary and not merely the sufficient. Hence for a particular attack, we want to identify the specific weaknesses upon which an attack depends in order to minimise the effort required to prevent it. Over-protective tagging schemes can unnecessarily increase message sizes, complexity and other communication overheads. Battery-powered embedded systems such as PDAs, cell phones, networked sensors and smart cards require such overheads to be minimal (Potlapally, Ravi, Raghunathan & Jha 2003). In these systems, minimising the resources needed to prevent potential attacks is clearly beneficial.

Generally, security protocol analysis tools have not been very good at finding type flaw attacks (Meadows 2003). Although more advanced tools exist (Meadows et al. 2004, Armando et al. 2005) that are capable of finding type flaw attacks, their analyses are based on high-level application layer specifications. However, by relying on application layer specifications, proofs of correctness may not be accurate with respect to the implementation of the protocol at the lower presentation layer. For example, attacks are sometimes found that are actually prevented by implicit presentation layer behaviour, and sometimes presentation layer attacks are completely overlooked. Furthermore, it is difficult to add detailed presentation layer corrections to application layer specifications, due to the level of abstraction exhibited at the application layer; often

corrections are presented informally or the entire design is changed. Once a type flaw attack has been discovered at the application layer, we need a way of verifying whether the attack is possible given the presentation layer specification, and to confirm the required presentation layer correction for inclusion in the formal specification.

In this paper we bridge the gap between the application and presentation layers by providing a framework for specifying security protocols for formal analyses at both levels of abstraction. Conveniently, we produce a single formal specification in which the presentation layer is transparent but easily reasoned with when required, thus improving readability and simplifying translation for input to off-the-shelf analysis tools. We demonstrate our approach by providing a multi-layered formal specification of the Group Domain of Interpretation (GDOI) protocol (Baughert, Hardjono, Harney & Weis 2001) in Object-Z (Duke & Rose 2000). The application layer model is analysed using the AVISPA model checking tool (Armando et al. 2005) to find two type flaw attacks. The attacks are then verified against the presentation layer model using Object-Z's schema calculus, and the particular presentation layer requirements required to prevent the attacks are formally derived for inclusion in the formal specification.

2 Related Work

In previous research Donovan et al. (1999) used CSP with the FDR model checker to analyse cryptographic protocols. They state that their approach is not good at finding type flaws. This is no doubt due to the lack of data structure support in the CSP model — message structure is captured in the event name.

Carlsen (1993) modified the logic of Communication, Knowledge and Time (CKT5) for analysis of cryptographic protocols, and demonstrated his approach by finding a type flaw attack on a version of the Neuman-Stubblebine protocol. The attack depends on confusion between a nonce¹ and a key. Carlsen found this attack by assuming that each item belonging to a particular type was distinct from items in all other types, and then by imposing a special requirement that nonces and keys were not necessarily distinct.

Bozzano and Delzanno (2002) used linear logic to discover that Millen's *ffgg* protocol (Millen 1999) is vulnerable to an attack which also requires one item to be interpreted as another. They assumed informally that the attack requires all items to have the same length. Bozzano (2002) used the same method to discover the type flaw attack on the Otway-Rees protocol. To allow for several items to be read as one, he introduced an extra concatenation operator *cons* to “glue together” different items in the message.

Theya et al. (1998) have used *strand spaces* to analyse cryptographic protocols. In this formalism, protocol correctness claims are expressed in terms of the connections between sequences of legitimate and/or intruder actions. Type flaw attacks are catered for by allowing unification of single items or pairs of items with other single items.

Cervesato (2001) used the strongly-typed Multi-set Rewriting (MSR) language which is based on term rewriting, linear logic and type theory, to express type flaw attacks. He uses polymorphism to allow for ‘confusable types’; the user of the approach decides which types are confusable in order to find type flaw attacks.

The attacks considered in these approaches generally involve simple ‘type confusion’, in which message

items of one type are confused with items of another type, or in the more advanced models, in which an item of one type is confused with the concatenation of two or more other items of varying types. For instance, Figure 1 shows two identical bit string representations of a message at the presentation layer consisting of an agent identifier and a nonce, and for each, an alternative way of interpreting the message at the application layer. The first illustrates the scenario in which the single nonce item is interpreted as a key, and the second illustrates the scenario in which the concatenation of the agent identifier with the nonce is interpreted as a key.

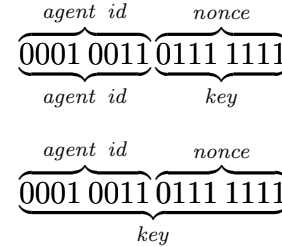


Figure 1: Misinterpretation of message items.

Meadows (2002) highlighted the further possibility of attacks in which sub-items of one type may be confused with sub-items of another type. This is particularly imaginable in the case where agents use different parsing algorithms for different protocols. In Figure 2, for instance, not only is the message interpreted at the application layer to consist of different types, but the item lengths are not even the same. Motivated by this, Meadows (2003) investigated a presentation layer procedure for determining the probability of any kind of type confusion occurring for any two pairs of protocol messages.

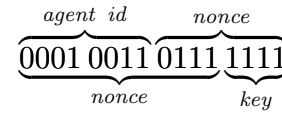


Figure 2: Misinterpretation of message sub-items.

Recently Meadows et al. (2004) analysed the GDOI protocol using the purpose-built NRL Protocol Analyzer to find a type flaw attack (Attack B) that takes advantage of the fact that GDOI is built on top of a protocol that does not tag message headers. Discovery of the attack also led to the manual discovery of a second type flaw attack (Attack A). Unfortunately, the tool could not have found Attack A since it lacked the ability to model associativity of message concatenation.

On the other hand, the AVISPA tool (Armando et al. 2005) handles associativity of message concatenation, so we were convinced it would successfully find both type flaw attacks on the GDOI protocol. Nevertheless, most protocol analysis tools (including AVISPA) are aimed at application layer models. Hence, it is difficult to include specific presentation layer requirements, for verification of the presentation layer specification.

Previously we demonstrated the value of using the Object-Z formal specification language for reasoning about the kind of type flaw attacks illustrated in Figures 1 and 2, enabling us to determine whether potential type flaw attacks are, or are not, actually possible for a given presentation layer specification of a security protocol (Long 2005). In this paper we bridge the

¹A nonce (Gong 1993) is a datum that is unique to each protocol instance.

1. $A \longrightarrow S : M, \{h(N_A, G), N_A, G\}_{K_{AS}}$
2. $S \longrightarrow A : M, \{h(N_A, N_S, SA), N_S, SA\}_{K_{AS}}$
3. $A \longrightarrow S : M, \{h(N_A, N_S, \{N_A, N_S\}_{K_A^{-1}}), \{N_A, N_S\}_{K_A^{-1}}\}_{K_{AS}}$
4. $S \longrightarrow A : M, \{h(N_A, N_S, Q, K, \{N_A, N_S\}_{K_S^{-1}}), Q, K, \{N_A, N_S\}_{K_S^{-1}}\}_{K_{AS}}$

Figure 3: Group Key Pull protocol.

gap between application and presentation layer verification by taking advantage of the AVISPA tool for automated analysis at the application layer followed by formal proof at the presentation layer.

3 The GDOI Protocol

The Group Domain of Interpretation (GDOI) protocol (Baughner et al. 2001) is proposed for group key distribution, in which each key is secret to a group of agents for secure communication amongst its members. Based on related work by Meadows et al. (2004), its two subprotocols relevant to our analysis are described below.

3.1 Group Key Pull Protocol

Agents wishing to join a group take part in the *Group Key Pull* protocol to learn the current group key (described using the standard notation in Figure 3).

In step 1, the new group member, Alice A , sends a message consisting of her nonce N_A , the name G of the group she wishes to join, and a hash of these items $h(N_A, G)$, all encrypted (denoted by ‘{ }’) using the key K_{AS} shared by her and the server, Sam S . A message header M is prepended to the message.

In step 2, Sam responds by sending Alice a message consisting of his nonce N_S , the security association SA , and a hash of these items accompanied by Alice’s nonce, again encrypted and prepended by the message header. The security association describes the security functions and policies used by the group.

In step 3, using her private key K_A^{-1} , Alice creates a signature from the two nonces. Her request for a key (message 3) contains this signature, and a hash of the signature accompanied by the original nonce values.

Like Alice, Sam creates a signature using his private key K_S^{-1} from the two nonces, and in step 4 sends a similar message to Alice containing the current sequence number Q , and the current group key K . (The sequence number must be known by all group members for use in the Group Key Push protocol.) Alice authenticates message 4 by ensuring the nonce values are the same as those she signed in message 3, and records the sequence number Q and key K for subsequent secure communication amongst the group.

3.2 Group Key Push Protocol

The group key is changed for the entire group by having all members act in Alice’s role for the *Group Key Push* protocol described in Figure 4.

5. $S \longrightarrow A : M, \{Q, SA, K^*, \{M, Q, SA, K^*\}_{K_S^{-1}}\}_K$

Figure 4: Group Key Push protocol.

In this single message protocol, Sam sends a message to Alice containing the sequence number, security association, and the new group key K^* , along with a signature taken over the entire content including the message header. The message is encrypted

using the current group key K . The sequence number is incremented by Sam each time the group key is changed so as to avoid the possibility of replay attacks on this protocol. Therefore, on receipt of this message, Alice checks that the sequence number is the one expected before accepting the key K^* as the new group key.

4 Specifying GDOI in Object-Z

Object-Z (Duke & Rose 2000) is an object-oriented formal specification language in which set theory and logic is used to describe the internal states of system classes and the behaviour of class operations on those states. In previous work (Long 2005) we presented Object-Z data types for modelling messages at the presentation layer. We reuse these structures for the foundation of this specification.

We assume a given set $ATOM$ of ‘atoms’ from which all messages are constructed at the presentation layer (for example, bits or bytes). Then the set of all messages MSG is the set of all possible sequences of atoms.

$$MSG == \text{seq } ATOM$$

Then we declare eight subsets of MSG for the different types of data items that exist at the application layer: header items HID , nonces NON , group identifiers GID , security associations SEC , sequence numbers SEQ , keys KEY , encrypted items ENS and hashed items HSH .

$$\begin{array}{l} HDR, NON : \mathbb{P} MSG \\ GID, SEC : \mathbb{P} MSG \\ SEQ, KEY : \mathbb{P} MSG \\ ENC, HSH : \mathbb{P} MSG \end{array}$$

The subsets are not necessarily disjoint which means that individual items may belong to one or more subsets of MSG . Keys are divided into another three subsets (symmetric keys SYM , public keys PUB and private keys PRV) enabling us to model both symmetric key and public key encryption.

$$\begin{array}{l} SYM : \mathbb{P} KEY \\ PUB : \mathbb{P} KEY \\ PRV : \mathbb{P} KEY \end{array}$$

To allow for analyses of type flaw attacks, our data structures let different agents interpret the same message as consisting of different sequences of typed items. However, when two agents ‘speak the same language’ or agree on the type structure of the message, there should be no ambiguity.

Instead of enforcing a specific correlation between the application and presentation layers, we assume the following global axiom which says that if two agents both interpret the initial part of a message to be a particular type of item, then the values they associate with this item are identical. (The sequence concatenation operator ‘ \frown ’ forms a single message from the two supplied messages.)

$$\begin{array}{|l} (\forall m, n : HDR; o, p : MSG \bullet \\ m \wedge o = n \wedge p \Rightarrow m = n) \end{array}$$

Thus, if two identical messages ' $m \wedge o$ ' and ' $n \wedge p$ ' begin with items m and n , both of which are interpreted to be of type *HDR*, then m and n must be the same message header. We have omitted the predicates constraining the other item types for the sake of brevity.

Our approach relies on unifying messages encrypted with the same key. To enable this, the encrypt function *enc* ensures that for every key there is a unique function that produces a unique encrypted item for each message.

$$\begin{array}{|l} enc : KEY \mapsto (MSG \mapsto ENC) \end{array}$$

The hash function *hsh* used by the Group Key Pull protocol takes a key as input. The following specification ensures that for every key there is a unique function that produces at most one corresponding hash value for each message.

$$\begin{array}{|l} hsh : KEY \mapsto (MSG \rightarrow HSH) \end{array}$$

Each public key is associated with a unique private key (its inverse) as specified by the function *inv*. In practice, this function is not one that would be globally available for use by protocol agents. However, it is convenient to have this association formalised. Then for the purpose of our specification we only need to allocate public keys to agents; private keys can be accessed using the *inv* function. In Section 5 we use a model checking tool for verification that provides a function *inv* associating public and private keys in the same way.

$$\begin{array}{|l} inv : PUB \mapsto PRV \end{array}$$

4.1 Protocol Roles

Although protocol descriptions refer to agents by name, in reality these protocols can be run between any pair of agents. In fact, when we refer to agent Alice, Alice is the name we are giving, not to the agent participating in the interaction, but to the role the agent is participating in. We could model the protocol as a single system or as the interaction of specific agents. However, as in other approaches (Sneekenes 1992, Ryan, Schneider, Goldsmith, Lowe & Roscoe 2000), we choose to model the protocol in terms of protocol roles and the interactions between them.

Between the two subprotocols, there is a total of four roles agents can play: *AlicePULL*, *SamPULL*, *AlicePUSH* and *SamPUSH*. Each role is captured within a single Object-Z class specification, including only information and operations relevant to the role it is modelling.

The first class *AlicePULL* corresponds to Alice's role in the Group Key Pull protocol. All items used by Alice in this role are declared as state variables in the state schema. Alice is required to generate a 'fresh' value for her nonce N_A and the message header M for each protocol instance. To allow for this, Alice keeps a record *used* of previously used items.

<i>AlicePULL</i>	$\begin{array}{l} M : HDR; N_A, N_S : NON; G : GID \\ Q : SEQ; SA : SEC; K_{AS}, K : SYM \\ K_A, K_S : PUB; used : \mathbb{P} MSG \end{array}$
<i>initiate</i>	$\begin{array}{l} \Delta(M, N_A, used) \\ msg! : MSG \\ \\ M' \notin used \wedge N_A' \notin used \\ used' = used \cup \{M', N_A'\} \\ msg! = M' \wedge enc(K_{AS})(hsh(K_{AS})(N_A' \wedge G) \wedge N_A' \wedge G) \end{array}$
<i>requestKey</i>	$\begin{array}{l} \Delta(N_S, SA) \\ msg?, msg! : MSG \\ \\ msg? = M \wedge enc(K_{AS})(hsh(K_{AS})(N_A \wedge N_S' \wedge SA') \wedge N_S' \wedge SA') \\ msg! = M \wedge enc(K_{AS})(hsh(K_{AS})(N_A \wedge N_S' \wedge enc(inv(K_A))(N_A \wedge N_S')) \wedge enc(inv(K_A))(N_A \wedge N_S')) \end{array}$
<i>pullKey</i>	$\begin{array}{l} \Delta(K, Q) \\ msg? : MSG \\ \\ msg? = M \wedge enc(K_{AS})(hsh(K_{AS})(N_A \wedge N_S \wedge Q' \wedge K' \wedge enc(inv(K_S))(N_A \wedge N_S)) \wedge Q' \wedge K' \wedge enc(inv(K_S))(N_A \wedge N_S)) \end{array}$

Within operation schemas, pre-state variables are undecorated and denote the value before execution of the operation, whereas post-state variables are decorated with a prime ' $'$ ' and denote the value after execution of the operation. The symbol ' Δ ' declares pre-state and post-state variables for each of the named variables, indicating that they may be changed by the operation. Incoming and outgoing messages are specified using input and output variables, denoted by '?' and '!' respectively.

Operation *initiate* corresponds to step 1 of the Group Key Pull protocol. Alice chooses fresh values for her nonce N_A' and the message header M' by checking that they do not yet belong to the set *used* of previously used values. She updates her record of used items to include these values using the set union operator. Appropriate functions are applied to the items required to construct message 1 and the output variable *msg!* is updated with the resultant message.

Operation *requestKey* corresponds to step 2. Use of post-state variables N_S' and SA' in the description of the incoming message *msg?* models the way Alice learns these values for future use. Use of the function *inv* accesses Alice's private key for creating her signature over the nonces.

Finally, *pullKey* corresponds to Alice receiving message 4 in which she learns the value of the key K' and the current sequence number Q' for the group.

The second class specified for the Group Key Pull protocol is for Sam's role. Operation *respond* corresponds to step 2. Like Alice, Sam ensures his nonce N_S' has not been used previously. On receiving Alice's nonce N_A' from the incoming message, he updates the message in transit with his fresh nonce and the security association SA . Operation *giveKey* corresponds to step 4 in which Sam receives Alice's request for the key in message 3 and sends the group key K and sequence number Q in message 4.

<i>Sam_{PULL}</i>
$M : HDR; N_A, N_S : NON; G : GID$ $Q : SEQ; SA : SEC; K_{AS}, K : SYM$ $K_S, K_A : PUB; used : \mathbb{P} MSG$
<i>respond</i>
$\Delta(M, N_S, N_A, used); msg?, msg! : MSG$ $N_S' \notin used \wedge used' = used \cup \{N_S'\}$ $msg? = M' \frown enc(K_{AS})(hsh(K_{AS})(N_A' \frown G) \frown N_A' \frown G)$ $msg! = M' \frown enc(K_{AS})(hsh(K_{AS})(N_A' \frown N_S' \frown SA) \frown N_S' \frown SA)$
<i>giveKey</i>
$msg?, msg! : MSG$ $msg? = M \frown enc(K_{AS})(hsh(K_{AS})(N_A \frown N_S \frown enc(inv(K_A))(N_A \frown N_S)) \frown enc(inv(K_A))(N_A \frown N_S))$ $msg! = M \frown enc(K_{AS})(hsh(K_{AS})(N_A \frown N_S \frown Q \frown K \frown enc(inv(K_S))(N_A \frown N_S)) \frown Q \frown K \frown enc(inv(K_S))(N_A \frown N_S))$

Sam's role in the Group Key Push protocol has one operation *pushKey*, in which he distributes the new group key K^* along with the current group sequence number Q and security association SA , all encrypted with the previous group key K . Group keys, sequence numbers and security associations are not renewed in each protocol instance since they must remain the same for each group member participating. So for our analyses, we assemble various scenarios by coordinating these values as required in the initialisation schema *Init* (in Section 6). Therefore, there is no need in either of Sam's roles to ensure they are fresh.

<i>Sam_{PUSH}</i>
$M : HDR; Q : SEQ; SA : SEC$ $K, K^* : SYM; K_S : PUB; used : \mathbb{P} MSG$
<i>pushKey</i>
$\Delta(M, used); msg! : MSG$ $M' \notin used \wedge used' = used \cup \{M'\}$ $msg! = M' \frown enc(K)(Q \frown SA \frown K^* \frown enc(inv(K_S))(M' \frown Q \frown SA \frown K^*))$

Finally, Alice has one operation *getKey* for receiving the Group Key Push message. Alice updates her value of the group key by using the post-state variable K' in the description of the incoming message. Earlier we mentioned that Alice checks the sequence number to avoid replay attacks. The value she expects is based on the value she received previously in the Group Key Pull protocol. However, at this level we are focusing on independent roles, hence the value she uses is hard-coded in the initialisation predicate when assembling particular scenarios we are interested in.

<i>Alice_{PUSH}</i>
$M : HDR; Q : SEQ; SA : SEC$ $K : SYM; K_S : PUB$
<i>getKey</i>
$\Delta(M, SA, K); msg? : MSG$ $msg? = M' \frown enc(K)(Q \frown SA' \frown K' \frown enc(inv(K_S))(M' \frown Q \frown SA' \frown K'))$

5 Modelling GDOI in AVISPA

AVISPA (Armando et al. 2005) is a tool for the Automated Validation of Internet Security-sensitive Protocols and Applications. Protocol roles are modelled in the High-Level Protocol Specification Language (HLPSP) as state transition systems in a similar way to our Object-Z specification, thus making the translation straightforward. The tool translates HLPSP specifications into an Intermediate Format (IF) describing a single infinite-state transition system for analysis by any of four back-end tools. Two of the back-end tools, the On-the-fly Model-Checker (OFMC) and the Constraint-Logic-based Attack Searcher (CL-AtSe), support type flaw detection.

As an example, the following HLPSP role *sam_{push}* corresponds to the Object-Z *Sam_{PUSH}* class. In Object-Z, we equate values used by multiple roles in particular scenarios in the initialisation schema. However, in AVISPA we equate such values by using parameters, and instantiating roles with particular values at the top-level *environment* role. Therefore, those values we wish to control are declared as parameters, whereas, values used within a single role only are declared as local variables. Declaring the channel with attribute 'dy' indicates the presence of a Dolev-Yao (Dolev & Yao 1983) intruder who has complete control over messages sent between agents.

```

role sam_push(
  A, S : agent, Q, SA : text,
  K, K2 : symmetric_key,
  Ks : public_key,
  MSG : channel(dy))
played_by S def=
local
  M : text
transition
  pushKey.
    MSG(start) = |>
    M' := new() /\
    MSG(M'.{Q.SA.K2.
      {M'.Q.SA.K2}_inv(Ks)}_K) /\
    witness(S,A,alice_server_k,K2)
end role

```

Operation *pushKey* corresponds to the Object-Z *pushKey* operation. In the Object-Z specification we ensured the new value of the message header was fresh by checking it was not in the set of *used* values. Conveniently, the HLPSP provides a special operator *new()* that achieves a similar effect.

The tool analyses authentication properties by attempting to match pairs of *witness* and *request* events. In the above operation, the witness event indicates that Sam intends to agree on key K2 with Alice. (A *protocol id* *alice_server_k* is used to identify the pair.) The matching request event is made by Alice in the following HLPSP role where she learns the new group key K'.

```

role alice_push(
  A, S : agent, Q : text,
  K : symmetric_key,
  Ks : public_key,
  MSG : channel(dy))
played_by A def=
local
  M, SA : text
transition
  getKey.
    MSG(M'.{Q.SA'.K'}).

```

```

    {M'.Q.SA'.K'}_inv(Ks)}_K) =|>
    request(A,S,alice_server_k,K')
end role

```

HLPSP provides a generic authentication property `authentication_on` which, given a protocol id, will ensure that the request event for that id is always preceded by the matching witness event. If any of the supporting tools find a trace in which the request event is not preceded by the corresponding witness event, an attack will have been found and will be presented to the user. Therefore, to ensure the value Alice accepts for the key is the same as the value Sam intended for the key, we state ‘`authentication_on alice_server_k`’ as the goal. AVISPA allows us to check more requirements, however, we require only this one to find the two type flaw attacks.

6 Verifying Attack A

Using the AVISPA model of the GDOI protocol, AVISPA’s CL-AtSe tool successfully finds the type flaw attack Meadows et al. (2004) were unable to find using the NRL Protocol Analyzer. The attack (Attack A shown in Figure 5) occurs in the scenario where the Group Key Pull protocol is run between a dishonest member, Carl, and the key server Sam, followed by the Group Key Push protocol between an honest member, Alice, and Carl who is posing as Sam.

Being a (dishonest) member of the group, we assume Carl already knows the current sequence number Q . Since all members of a group have the means to calculate the next sequence number for identifying replay attacks on the Group Key Push protocol, he can produce the next sequence number Q^* that the group members are expecting to accompany the new group key.

Carl initiates the Group Key Pull protocol using as his nonce N_C , the concatenation of a message header M^* , the next value of the sequence number Q^* and a security association SA^* . Interpreting the composition of these fields as Carl’s nonce, Sam responds appropriately with his nonce N_S and the security association SA in step 2. Continuing the protocol, Carl sends message 3 in which the fake nonce and Sam’s nonce are signed with his private key. In step 4 Sam sends the current sequence number Q and group key K to Carl, together with his own signature over Carl’s fake nonce and Sam’s nonce.

Interestingly, if this signature is sent to Alice as part of a Group Key Push message, she may interpret the concatenation of values M^* , Q^* , and SA^* , that Carl chose for his nonce, hence interpreting N_S as the new group key. Knowing this, Carl initiates an instance of this protocol (step 5) with Alice using Sam’s signature from the previous protocol. Thus, Carl will have successfully masqueraded as the trusted group key server and tricked Alice into believing that Sam’s nonce is the new group key. Carl can initiate the Group Key Push protocol with all members of the group resulting in group communication with an unauthorised key. Furthermore, future communication amongst the group will be completely disrupted for new members and for the group key server.

When this scenario is executed in the AVISPA tool, on accepting Sam’s nonce as the new group key, Alice will signal a `request` event for the value N_S . However, since there is no corresponding `witness` event for this value, the CL-AtSe attack searcher will know that the value Alice received for the key is not the correct one and that the authentication goal has been violated.

In order to formally prove the existence of this attack at the presentation layer and to identify the

specific requirements that suppress it, we first derive a specification of Carl’s involvement in both instances as follows.

<i>Carl</i>	
	$M, M^* : HDR; N_S : NON; G : GID$ $Q, Q^* : SEQ; SA, SA^* : SEC$ $K_{AS}, K : SYM; K_A : PUB$
<i>initiate</i>	$msg! : MSG$ $msg! = M \wedge enc(K_{AS})(hsh(K_{AS})(M^* \wedge Q^* \wedge SA^* \wedge G) \wedge M^* \wedge Q^* \wedge SA^* \wedge G)$
<i>requestKey</i>	$\Delta(N_S, SA)$ $msg?, msg! : MSG$ $msg? = M \wedge enc(K_{AS})(hsh(K_{AS})(M^* \wedge Q^* \wedge SA^* \wedge N_S' \wedge SA') \wedge N_S' \wedge SA')$ $msg! = M \wedge enc(K_{AS})(hsh(K_{AS})(M^* \wedge Q^* \wedge SA^* \wedge N_S' \wedge enc(inv(K_A))(M^* \wedge Q^* \wedge SA^* \wedge N_S')) \wedge enc(inv(K_A))(M^* \wedge Q^* \wedge SA^* \wedge N_S'))$
<i>pushKey</i>	$\Delta(K, Q)$ $msg?, msg! : MSG$ $(\exists sig : ENC \bullet$ $msg? = M \wedge enc(K_{AS})(hsh(K_{AS})(M^* \wedge Q^* \wedge SA^* \wedge N_S \wedge Q' \wedge K' \wedge sig) \wedge Q' \wedge K' \wedge sig) \wedge$ $msg! = M^* \wedge enc(K')(Q^* \wedge SA^* \wedge N_S \wedge sig))$

The first two operations, *initiate* and *requestKey*, are similar to the operations of the same name in Alice’s role for the Group Key Pull protocol, only the nonce N_A is replaced by the concatenation of values, M^* , Q^* and SA^* . In operation *pushKey*, Carl uses the signature *sig*, created by the server in response to his dummy request, to create the fake Group Key Push message.

An instance of each role required to simulate the scenario in which the attack is present is declared below.

```

carl : Carl; sam_L : SamPULL
sam_S : SamPUSH; alice : AlicePUSH

```

First we specify the initial conditions that must hold for the attack to be possible in the following initialisation schema *Init*. We identify which variables belong to each agent instance by prefixing them with the instance name.

Initially, agents will have the same values for certain state variables such as keys, to ensure they can communicate successfully. (The group identifier and sequence number used are also coordinated in this way.) This information is specified in the initialisation schema *Init*.

```

Init
carl.G = sam_L.G ∧ carl.K_AS = sam_L.K_AS
sam_L.K_S = alice.K_S ∧ carl.K_A = sam_L.K_A
carl.Q* = alice.Q ∧ sam_L.K = alice.K

```

Using Object-Z’s sequential composition operator ‘*;*’, the single operation *attack_A* specifies the complete

1. $C \longrightarrow S : M, \{h(N_C, G), N_C, G\}_{K_{CS}} (N_C = M^*, Q^*, SA^*)$
2. $S \longrightarrow C : M, \{h(N_C, N_S, SA), N_S, SA\}_{K_{CS}}$
3. $C \longrightarrow S : M, \{h(N_C, N_S, \{N_C, N_S\}_{K_C^{-1}}), \{N_C, N_S\}_{K_C^{-1}}\}_{K_{CS}}$
4. $S \longrightarrow C : M, \{h(N_C, N_S, Q, K, \{N_C, N_S\}_{K_S^{-1}}), Q, K, \{N_C, N_S\}_{K_S^{-1}}\}_{K_{CS}}$
5. $C(S) \longrightarrow A : M^*, \{Q^*, SA^*, N_S, \{M^*, Q^*, SA^*, N_S\}_{K_S^{-1}}\}_K$

Figure 5: Attack A on the GDOI protocol.

sequence of operations that lead to the attack after initialisation.

$$\text{attack}_A \triangleq \text{Init} \wedge \text{carl.initiate} \circledast \\ \text{sam}_L.\text{respond} \circledast \text{carl.requestKey} \circledast \\ \text{sam}_L.\text{giveKey} \circledast \text{carl.pushKey} \circledast \\ \text{alice.getKey}$$

Sequential composition is achieved by taking the union of the variables in the Δ -list and the conjunction of both operation predicates, where we assume the existence of an *intermediate state* in which the primed variables from the first operation are equated with the unprimed variables of the same name in the second operation. An intermediate state is introduced by declaring an *intermediate variable* (identified by a double-prime) for each state variable. Additionally, those output variables from the first operation with the same base name as input variables of the second operation are equated and hidden. We do this by introducing an existentially quantified variable to replace them in the same way as we do for intermediate state variables. For example, composition of Carl's *initiate* operation with *sam_L.respond* produces an intermediate variable *msg''* as follows.

$$(\exists \text{msg}' : \text{MSG} \bullet \\ \text{msg}'' = \text{carl}.M \wedge \\ \text{enc}(\text{carl}.K_{AS})(\text{hsh}(\text{carl}.K_{AS})(\text{carl}.M^* \wedge \\ \text{carl}.Q^* \wedge \text{carl}.SA^* \wedge \text{carl}.G) \wedge \text{carl}.M^* \wedge \\ \text{carl}.Q^* \wedge \text{carl}.SA^* \wedge \text{carl}.G) \wedge \\ \text{msg}'' = \text{sam}_L.M' \wedge \\ \text{enc}(\text{sam}_L.K_{AS})(\text{hsh}(\text{sam}_L.K_{AS})(\text{sam}_L.N_A' \wedge \\ \text{sam}_L.G) \wedge \text{sam}_L.N_A' \wedge \text{sam}_L.G))$$

Since we know $\text{carl}.G = \text{sam}_L.G$ and $\text{carl}.K_{AS} = \text{sam}_L.K_{AS}$ from initialisation, and due to the constraints placed on presentation layer data types in Section 4, we can derive the equalities $\text{sam}_L.M' = \text{carl}.M$ and (more importantly) $\text{sam}_L.N_A' = \text{carl}.M^* \wedge \text{carl}.Q^* \wedge \text{carl}.SA^*$, allowing us to reason about type flaw attacks in more detail.

Composition of the entire sequence of operations defining the attack trace attack_A results in the following schema.

attack_A $\Delta(\text{sam}_L.M, \text{sam}_L.N_S, \text{sam}_L.N_A, \text{alice}.SA$ $\text{sam}_L.\text{used}, \text{carl}.N_S, \text{carl}.SA,$ $\text{carl}.K, \text{carl}.Q, \text{alice}.M, \text{alice}.K)$
$\text{carl}.G = \text{sam}_L.G \wedge \text{carl}.K_{AS} = \text{sam}_L.K_{AS}$ $\text{carl}.K_A = \text{sam}_L.K_A \wedge \text{carl}.Q^* = \text{alice}.Q$ $\text{sam}_L.K = \text{alice}.K \wedge \text{sam}_L.K_S = \text{alice}.K_S$ $\text{carl}.SA' = \text{sam}_L.SA \wedge \text{carl}.N_S' = \text{sam}_L.N_S'$ $\text{sam}_L.N_S' \notin \text{sam}_L.\text{used}$ $\text{sam}_L.\text{used}' = \text{sam}_L.\text{used} \cup \{\text{sam}_L.N_S'\}$ $\text{sam}_L.M' = \text{carl}.M \wedge \text{alice}.K' = \text{sam}_L.N_S'$ $\text{sam}_L.N_A' = \text{carl}.M^* \wedge \text{carl}.Q^* \wedge \text{carl}.SA^*$ $\text{carl}.K' = \text{sam}_L.K \wedge \text{carl}.Q' = \text{sam}_L.Q$ $\text{alice}.M' = \text{carl}.M^* \wedge \text{alice}.SA' = \text{carl}.SA^*$

Based on the authentication requirement analysed by the tool, a suitable requirement for our verification is that by the end of the Group Key Push protocol, the values that Sam and Alice have for the new group key are identical. Therefore, insecurity of this protocol is proven by demonstrating that after the attack, these values are not equal.

$$\text{attack}_A \Rightarrow \text{sam}_S.K'^* \neq \text{alice}.K'$$

We know that after the attack, Alice's value for the group key is equal to the value of Sam's nonce ($\text{alice}.K' = \text{sam}_L.N_S'$). It is reasonable to assume that the value for Sam's nonce is not equal to the value Sam has for the new group key ($\text{sam}_L.N_S' \neq \text{sam}_S.K^*$), and with this assumption we can conclude that the above predicate holds, thus verifying insecurity of the protocol.

The attack operation also reveals the following two conditions that must be preserved by the presentation layer for the attack to succeed.

$$(\exists \text{sam}_L.N_A' : \text{NON} \bullet \\ \text{sam}_L.N_A' = \text{carl}.M^* \wedge \text{carl}.Q^* \wedge \text{carl}.SA^*) \\ (\exists \text{alice}.K' : \text{KEY}; \text{sam}_L.N_S' : \text{NON} \bullet \\ \text{alice}.K' = \text{sam}_L.N_S')$$

The first condition states that a nonce can be constructed from items $\text{carl}.M^*$, $\text{carl}.Q^*$ and $\text{carl}.SA^*$. If this condition is not met, Sam will not accept message 1 from Carl and the protocol will be aborted. The second condition states that Alice can interpret a nonce as a key. If this condition is not met, she will not accept message 5 and again the protocol will be aborted.

These results are somewhat expected; however, we now know for certain that the attack will be prevented by avoiding at least one of these conditions. Thus, the negation of these conditions forms a special presentation layer requirement, providing an Object-Z specification of the protocol secure against this attack, without having to make any additional changes to the application layer design. Either of the following two conditions must hold to prevent the attack.

$$\neg(\exists \text{sam}_L.N_A' : \text{NON} \bullet \\ \text{sam}_L.N_A' = \text{carl}.M^* \wedge \text{carl}.Q^* \wedge \text{carl}.SA^*) \\ \neg(\exists \text{alice}.K' : \text{KEY}; \text{sam}_L.N_S' : \text{NON} \bullet \\ \text{alice}.K' = \text{sam}_L.N_S')$$

7 Verifying Attack B

The CL-AtSe tool also successfully finds the type flaw attack (Attack B shown in Figure 6) Meadows et al. (2004) found using the NRL Protocol Analyzer, this time in the scenario where the intruder, Dan *D*, has discovered the key K_{AS} shared between Alice and Sam, and hence also the group key K and sequence number. Additionally, it assumes Alice may play the

part of both a member and key server of the same group. Firstly, the Group Key Pull protocol is run between member Alice and Dan, whom she believes is the key server, Sam. Then the Group Key Push protocol is run between Dan (posing as Alice in her key server role) and group member Bob.

Alice initiates a Group Key Pull protocol with the key server, Sam. However, Dan intercepts message 1 and impersonates Sam by sending message 2 to Alice using the concatenation of the next value of the sequence number Q^* , a security association SA^* , and a key K^* as the value for the nonce N_D . Following the protocol, Alice creates a signature from her nonce and the value she believes is Sam's nonce. Once again, this signature could be passed off as the signature used in the Group Key Push protocol, this time where Alice's nonce is interpreted as the message header. Dan can use this signature and the group key he already knows to create a Group Key Push message (step 5). Thus, Dan will have successfully tricked Alice into accepting his chosen key K^* for secure communication amongst the group. Formalisation of Dan's behaviour follows.

<i>Dan</i>
$M : HDR; N_A : NON; G : GID; Q^* : SEQ$ $SA, SA^* : SEC; K_{AS}, K, K^* : SYM$
<i>respond</i>
$\Delta(M, N_A)$ $msg?, msg! : MSG$
$msg? = M' \wedge enc(K_{AS})(hsh(K_{AS})(N_A' \wedge G) \wedge N_A' \wedge G)$ $msg! = M' \wedge enc(K_{AS})(hsh(K_{AS})(N_A' \wedge Q^* \wedge SA^* \wedge K^* \wedge SA) \wedge Q^* \wedge SA^* \wedge K^* \wedge SA)$
<i>pushKey</i>
$msg?, msg! : MSG$
$(\exists sig : ENC \bullet$ $msg? = M \wedge enc(K_{AS})(hsh(K_{AS})(N_A \wedge Q^* \wedge SA^* \wedge K^* \wedge sig) \wedge sig)$ $msg! = N_A \wedge enc(K)(Q^* \wedge SA^* \wedge K^* \wedge sig))$

Operation *respond* is similar to the operation of the same name in Sam's role where Sam's nonce is replaced by the concatenation of the three decorated values chosen by Dan. In operation *pushKey*, Dan accepts Alice's request for the key and sends a Group Key Push message using the signature *sig* created by Alice in order to pose as her.

Once again, the required roles are coordinated to simulate the scenario in which the attack is found. Alice is playing the part of both a group member in the Group Key Pull protocol $alice_L : Alice_{PULL}$ and a server in the Group Key Push protocol $alice_S : Sam_{PUSH}$.

$$alice_L : Alice_{PULL} \wedge dan : Dan$$

$$bob : Alice_{PUSH} \wedge alice_S : Sam_{PUSH}$$

The initialisation predicate is specified below coordinating state variables belonging to each of the participating roles. Since the attack depends on Dan having discovered the key K_{AS} shared between Alice and Sam, we specify that $dan.K_{AS} = alice_L.K_{AS}$. Additionally, Dan knows the group for which Alice is establishing a key ($dan.G = alice_L.G$) and the key K used for the group ($dan.K = bob.K$). He also knows the current sequence number which means that the next value Q^* of the sequence number is the one

Bob is expecting in the Group Key Push protocol ($dan.Q^* = bob.Q$). Alice plays the role of the server in the Group Key Push protocol with Bob which means that her public key is the key Bob has for the server ($alice_S.K_A = bob.K_S$). It is also important to state that Alice's public key is the same in both of her roles ($alice_L.K_A = alice_S.K_A$).

<i>Init</i>
$dan.G = alice_L.G \wedge dan.K_{AS} = alice_L.K_{AS}$ $alice_L.K_A = alice_S.K_A \wedge dan.Q^* = bob.Q$ $dan.K = bob.K \wedge alice_L.K_A = bob.K_S$

We simulate the attack sequence by evaluating the sequential composition of the operations leading to the attack after initialisation.

$$attack_B \hat{=} Init \wedge alice_L.initiate \circ dan.respond \circ$$

$$alice_L.requestKey \circ dan.pushKey \circ$$

$$bob.getKey$$

Composition of the operations defining the attack trace $attack_B$ results in the following schema.

<i>attack_B</i>
$\Delta(alice_L.M, alice_L.N_A, alice_L.used, alice_L.N_S,$ $alice_L.SA, dan.M, dan.N_A, bob.M, bob.SA,$ $bob.K)$
$dan.G = alice_L.G \wedge dan.K_{AS} = alice_L.K_{AS}$ $dan.N_A' = alice_L.N_A' \wedge dan.Q^* = bob.Q$ $dan.K = bob.K \wedge alice_L.K_A = bob.K_S$ $alice_L.K_A = alice_S.K_A \wedge dan.M' = alice_L.M'$ $\{alice_L.M', alice_L.N_A'\} \cap alice_L.used = \emptyset$ $alice_L.used' = alice_L.used \cup \{alice_L.M', alice_L.N_A'\}$ $alice_L.N_S' = dan.Q^* \wedge dan.SA^* \wedge dan.K^*$ $alice_L.SA = dan.SA \wedge bob.M' = alice_L.N_A'$ $bob.SA' = dan.SA^* \wedge bob.K' = dan.K^*$

This time, insecurity of this protocol is proven by demonstrating that after the attack, the value Bob has for the new group key is not identical to the one distributed by Alice in her server role.

$$attack_B \Rightarrow alice_S.K^* \neq bob.K'$$

From the equalities formed in $attack_B$, we know that Bob's value of the new key is the same as the value Dan has chosen for the new group key ($bob.K' = dan.K^*$). It is reasonable to assume that the value he chose is not equivalent to the value Alice had chosen for the new group key ($alice_S.K^* \neq dan.K^*$). With this additional assumption the theorem above is true. Hence, we have verified Attack B.

Again we notice two conditions in the attack schema that must be preserved by the presentation layer for Attack B to succeed, and again, negating these produces the following two preventative conditions, one of which must hold to secure the protocol against the attack.

$$\neg(\exists alice_L.N_S' : NON \bullet$$

$$dan.Q^* \wedge dan.SA^* \wedge dan.K^*)$$

$$\neg(\exists bob.M' : HDR; dan.N_A' : NON \bullet$$

$$bob.M' = dan.N_A')$$

In practice, the GDOI specification now contains an informal presentation layer requirement that each signed message must contain a single tag, identifying whether it belongs to the Group Key Pull or Group Key Push protocol (Meadows et al. 2004). This is formally specified in the following predicate ensuring the

1. $A \longrightarrow D(S) : M, \{h(N_A, G), N_A, G\}_{K_{AS}}$
2. $D(S) \longrightarrow A : M, \{h(N_A, N_D, SA), N_D, SA\}_{K_{AS}} (N_D = Q^*, SA^*, K^*)$
3. $A \longrightarrow D(S) : M, \{h(N_A, N_D, \{N_A, N_D\}_{K_A^{-1}}), \{N_A, N_D\}_{K_A^{-1}}\}_{K_{AS}}$
5. $D(A) \longrightarrow B : N_A, \{Q^*, SA^*, K^*, \{N_A, Q^*, SA^*, K^*\}_{K_A^{-1}}\}_K$

Figure 6: Attack B on the GDOI protocol.

content type of the two signatures is distinguishable for all possible combinations of items. We know from our formal proofs that this requirement successfully secures the protocol from the two type flaw attacks since it implies both of the derived requirements.

$$\begin{aligned}
 &(\forall N_A, N_S : NON; M : HDR; Q : SEQ; \\
 &SA : SEC; K : KEY \bullet \\
 &N_A \frown N_S \neq M \frown Q \frown SA \frown K)
 \end{aligned}$$

8 Conclusion

In this paper we provided a formal specification of the Group Domain of Interpretation protocol using Object-Z, in which both the application and presentation layers were present. We were required to specify presentation layer constraints on the data types, allowing agents to interpret messages correctly when they agree on the type structure of the message, but allowing multiple potential interpretations otherwise. Since the two models of differing detail are encapsulated within the one specification, there was no need for complex transformations from one to the other.

We combined the relative strengths of model checking and theorem proving to verify the specification and to secure it against two type flaw attacks. The application layer model was analysed using the AVISPA model checking tool to find the attacks. Subsequently, we verified the attacks against the presentation layer model using the Object-Z schema calculus, confirming the assumptions that allow the attacks and deriving the presentation layer requirements that prevent them. Furthermore, we confirmed that the approach taken by GDOI to distinguish between the content of signatures is sufficient to secure the protocol against both attacks, independent of other tagging schemes used.

Acknowledgements

We would like to thank Catherine Meadows for help in understanding the GDOI protocol and the anonymous referees for their comments. This research was supported in part by the Defence Signals Directorate and the Australian Research Council via Linkage-Projects Grant LP0347620, Formally-Based Security Evaluation Procedures.

References

- Abadi, M. (2000), Security protocols and their properties, in 'Foundations of Secure Computation', NATO Science Series, IOS Press, pp. 39–60.
- Armando et al. (2005), The AVISPA tool for the automated validation of internet security protocols and applications, in 'Proceedings of the 17th International Conference on Computer-Aided Verification (CAV'05)', Vol. 3576 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 281–285.
- Baughner, M., Hardjono, T., Harney, H. & Weis, B. (2001), 'Group domain of interpretation for ISAKMP'. Archived at <http://www.watersprings.org/pub/id/draft-irtf-smug-gdoi-01.txt>, January 2001.
- Boyd, C. (1990), 'Hidden assumptions in cryptographic protocols', *IEEE Proceedings, Part E* pp. 433–436.
- Boyd, C. & Mao, W. (1993), On a limitation of BAN logic, in T. Helleseht, ed., 'Advances in Cryptology (EUROCRYPT'93)', Vol. 1055 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 240–246.
- Bozzano, M. (2002), A Logic-Based Approach to Model Checking of Parameterized and Infinite-State Systems, PhD thesis, DISI, University of Genova. <http://www.disi.unige.it/person/BozzanoM/publications.html>.
- Bozzano, M. & Delzanno, G. (2002), Automated protocol verification in linear logic, in 'Proceedings of the Fourth ACM SIGPLAN Conference on Principles and Practice of Declarative Programming', ACM Press, pp. 38–49.
- Carlsen, U. (1993), Using logics to detect implementation-dependent flaws, in 'Proceedings of the Ninth Annual Computer Security Applications Conference', IEEE Computer Society Press, pp. 64–73.
- Carlsen, U. (1994), Generating formal cryptographic protocol specifications, in 'Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy', IEEE Computer Society Press, pp. 137–146.
- Cervesato, I. (2001), 'Expressing type-flaw attacks in a strongly typed language'. Second Workshop on Foundations for Secure/Survivable Systems and Networks, Tokyo, Japan, 27 October 2001.
- Dolev, D. & Yao, A. C. (1983), 'On the security of public key protocols', *IEEE Transactions on Information Theory* **29**(2), 198–208.
- Donovan, B., Norris, P. & Lowe, G. (1999), Analyzing a library of security protocols using Casper and FDR, in 'Proceedings of the Workshop on Formal Methods and Security Protocols', Trento, Italy.
- Duke, R. & Rose, G. (2000), *Formal Object-Oriented Specification Using Object-Z*, Cornerstones of Computing, Macmillan Press Limited, UK.
- Gollmann, D. (2003), Analysing security protocols, in 'Formal Aspects of Security', Vol. 2629 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 71–80.
- Gong, L. (1993), Variations on the themes of message freshness and replay — or the difficulty of devising formal methods to analyze cryptographic

- protocols, *in* 'Proceedings of the Computer Security Foundations Workshop VI', IEEE Computer Society Press, pp. 131–136.
- Heather, J., Lowe, G. & Schneider, S. (2000), How to prevent type flaw attacks on security protocols, *in* 'Proceedings of 13th IEEE Computer Security Foundations Workshop (CSFW'00)', IEEE Computer Society Press, pp. 32–43.
- Long, B. W. (2005), Formal verification of a type flaw attack on a security protocol using Object-Z, *in* '4th International Conference of B and Z Users, ZB 2005', Vol. 3455 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 319–333.
- Meadows, C. (2002), Identifying potential type confusion in authenticated messages, *in* 'Proceedings of Workshop on Foundation of Computer Security (FCS'02)', pp. 75–84. Published as a joint DIKU technical report, <http://www.diku.dk/>.
- Meadows, C. (2003), A procedure for verifying security against type confusion attacks, *in* 'Proceedings of 16th IEEE Computer Security Foundations Workshop', IEEE Computer Society Press, pp. 62–72.
- Meadows, C., Syverson, P. & Cervesato, I. (2004), 'Formal specification and analysis of the Group Domain of Interpretation Protocol using NPA-TRL and the NRL Protocol Analyzer', *Journal of Computer Security* **12**(6), 893–931.
- Millen, J. K. (1999), A necessarily parallel attack, *in* 'Proceedings of the Workshop on Formal Methods and Security Protocols', Trento, Italy. <http://www.cs.bell-labs.com/who/nch/fmsp99/program.html>.
- Potlapally, N. R., Ravi, S., Raghunathan, A. & Jha, N. K. (2003), Analyzing the energy consumption of security protocols, *in* 'Proceedings of the 2003 international symposium on Low power electronics and design', ACM Press, pp. 30–35.
- Ryan, P., Schneider, S., Goldsmith, M., Lowe, G. & Roscoe, B. (2000), *The Modelling and Analysis of Security Protocols: The CSP Approach*, Addison-Wesley.
- Snekkenes, E. (1992), Roles in cryptographic protocols, *in* 'Proceedings of the 1992 IEEE Symposium on Research in Security and Privacy', IEEE Computer Society Press, pp. 105–119.
- Thayer Fábrega, F. J., Herzog, J. C. & Guttman, J. D. (1998), Strand spaces: Why is a security protocol correct?, *in* 'Proceedings of the 1998 IEEE Symposium on Security and Privacy', IEEE Computer Society Press, pp. 160–171.
- The Common Criteria Project Sponsoring Organisations (1999), *Common Criteria for Information Technology Security Evaluation*, 2.1 edn. ISO/IEC Standard 15408.

Access Control Models and Security Labelling

Chuchang Liu

Angela Billard

Maris Ozols

Nikifor Jeremic

Information Networks Division
Defence Science and Technology Organisation
PO Box 1500, Edinburgh
SA 5111, Australia

{*Chuchang.Liu, Angela.Billard, Maris.Ozols, Nikifor.Jeremic*}@dsto.defence.gov.au

Abstract

Security labels convey information that is utilised to perform access control decisions, specify protective measures, and aid in the determination of additional handling restrictions required by security policies. In discussing security labelling, one of the most important aspects is to investigate access control models and obtain an appropriate technique for specifying the kind of security policies that are required. One problem with previous approaches to the specification of access control policies is that they are based on an idealisation of the real problem and give a first approximation: may or may not a subject access a given object? The binary, logical function is the essential starting point, but is generally insufficient to guide the hard decisions that are required by a variety of applications in the real world. Focusing on the issues regarding security labelling, this paper first proposes a technique for expressing need-to-know policies that are regarded as the basis for security labelling and should be followed in the labelling process. Then, based on the proposed lattice access control model dealing with both security levels and categories of objects, several security labelling principles are given. Finally, we propose a dynamic model for security labelling that not only provides support for dynamic labelling within a system but also a functional base for the design and implementation of a security labelling system.

Keywords: open system, security label, access control, security policy, dynamic labelling model, assurance.

1 Introduction

Information is widely recognised as a valuable asset, even a commodity, in the current 'Information Age'. However, the value of information to its possessor is inextricably linked to the ability to protect and share it as the possessor deems appropriate. Information security includes the set of measures taken to protect data from unauthorised or accidental modification, destruction, or disclosure. Information security can be applied to all systems, whether automated, paper-based, human-based, or a combination of these. The systems considered in this paper are those that may be classed as open systems. An open system is viewed as a set of one or more computers, associated with software, peripherals, terminals, human operators, physical processes, information transfer methods, *etc.*, that forms an autonomous whole capable of storing, processing and transferring information.

Security labelling is one of the mechanisms that can be used in the provision of information security. It supports the correct handling of data within and

between systems, according to the sensitivity of the data. Typically, the value and sensitivity of information objects can vary greatly. Some objects contain critical information that requires a very high degree of protection, while others hardly require more than a very modest degree of protection. Furthermore, while objects may have an equal degree of sensitivity, the nature of these sensitivities may differ. The purpose of security labelling is to provide the means for making a concise assertion about an information object's sensitivities, which may be used with security policy to usefully determine the appropriate protection and handling. Information contained within security labels can be utilised to control access, specify protective measures, and determine additional handling restrictions required by communication security policies (Internet CIPSO Working Group 1993).

The earliest work regarding security labels can be traced back to (Weissman 1969), where the label of an object is used to reflect the classification of the object. Housley (1993) discussed a security labelling framework designed especially for the Internet; and more recently Liu and Orgun propose a checkable model for security labelling (Liu & Orgun 2006), which captures a variety of security requirements. However, the previously proposed model is a static model, and does not address dynamic changes to security labels. Therefore, a motivation of this work is to investigate the issues regarding the influence of dynamic factors, and the intention is to provide a dynamic model for security labelling. The notion of security labelling also appears in security-typed languages, which have been proposed recently to enforce security properties including confidentiality and integrity by type checking (Heintze & Riecke 1998, Myers & Liskov 2000, Zdancewic *et al.* 2001). In security-typed languages, types are extended with security labels to enforce information flow control, but these labels are usually applied only to denote security classes associated with users and the resources that programs access (Zheng & Myers 2004).

Security labels should reflect and satisfy the range of requirements specified in the policy for the system. Security policies are varied, and include policies for data integrity and data confidentiality. Biba (1977) proposed a model that specifies the rule-based controls for writing and deleting that are necessary to preserve data integrity. It also specifies rule-based controls for reading to prevent a high integrity process from relying on data that has less integrity than the process. With regard to confidentiality, Bell and LaPadula (1976) defined a model that specifies the rule-based controls for reading that are necessary to preserve data confidentiality, and it also specifies rule-based controls for writing to ensure that data is not copied to a container where confidentiality can not be

guaranteed. In both the Biba and the Bell-LaPadula models, the security label is a static attribute of the data.

In discussing security labelling, one of the most important aspects is to investigate access control models and obtain an appropriate technique for specifying the kind of security policies that need to be applied. The problem is to find an appropriate model for security labelling. One problem with previous approaches to specifying access control policy is that they are based on an idealisation of the true problem and give a first approximation: may or may not a subject access a given object? The binary, logical function is the essential starting point, but is generally insufficient to guide the hard decisions that are required in implementation. In order to provide evaluative criteria by which adequacy of security architectures may be assessed and compared, new approaches for describing the full access control requirements are demanded.

Although there is a standard for security labelling defined in (FIPS188 1994) specifically for the purposes of information transfers, there is a lack of formal techniques for modelling security labelling. Therefore, one of the motivations of our work is to discuss these issues in general and provide a formal methodology for modelling security labels. We do not intend to describe specific practical labelling systems, but the methods and techniques of modelling security labelling would support the design and implementation of such systems.

One major contribution of this paper is to propose a lattice-based access control model for the support of security labelling. Focusing on the issues regarding security labelling, this paper first proposes a technique to express need-to-know policies that are regarded as a basis for security labelling and should be followed in the labelling process. Then, based on the proposed lattice access control model dealing with both security levels and categories of objects, we propose several security labelling principles. The other major contribution of the paper is that we propose a dynamic model for security labelling. This model not only provides support for dynamic labelling within a system but also provides a functional base for the design and implementation of a security labelling system. The main difference between the dynamic model and the checkable model proposed in (Liu & Orgun 2006) is that the dynamic model is capable of dealing with the dynamics of a system, including changes to a policy, and provides support for relabelling. In contrast, the checkable model is static and has no such ability although it captures a variety of security requirements in the labelling process.

The structure of the paper is as follows. Section 2 discusses access control models, and proposes a formal approach to specifying need-to-know policies. Section 3 presents several security labelling principles based on the lattice access control model. Section 4 discusses the security labelling framework, and proposes a model for security labelling. Section 5 discusses implementation of this model, and Section 6 discusses other issues, including the dynamic aspects of our security labelling model and further considerations in relation to assurance. Section 7 concludes the paper.

2 Access Control Models

A security policy is a set of criteria for the provision of security services that defines and constrains the activities of data processing facilities within a system; it states strict requirements for how material is to

be handled during storage, processing and transmission in order to maintain the security conditions that should be satisfied. With regard to access control, a security policy in its simplest form is a subset of a system-level security policy that defines the means for enforcing the access control policy (Ferraiolo, Kuhn & Chandramouli 2003).

Security labels provide information for making access control decisions within a security system, while the access control policy is the foundation on which we base security labelling. In this section, we firstly discuss how to express an access control policy. We then present a formal technique for specifying a need-to-know policy, which combines the security classification and the security categories of objects, and propose a practical access control model based on the previous lattice-representation method proposed by Sandhu (1993).

2.1 Subjects and Objects

First let us define a few terms.

Subjects are active entities that can perform actions within a system. Typically, we consider subjects to be processes executing on behalf of users. Users may also be considered to be subjects; however, because users may run many processes concurrently, when a user is treated as a subject, it is in association with a single process. For a given system, there is a set of subjects that is expressed as:

$$\mathcal{S} = \{s_i \mid i = 0, 1 \dots, n\}.$$

Objects are passive entities that can undergo actions. Typical examples of objects are resources, tools, or mechanisms used to maintain a condition of security in a computerized environment. Similarly, for the given system, we also have a set of objects expressed as:

$$\mathcal{O} = \{o_j \mid j = 0, 1 \dots, m\}.$$

In terms of access control, we say that the entity requesting access to a resource is the subject of the access and the resource that the subject attempts to access is the object of the access.

We now define a relation and an operation over the object set \mathcal{O} , respectively, as follows:

- If object o is a part of object o^+ or o^+ is a new object generated by adding something (data information) to object o , then we say that o^+ contains o or o is contained in o^+ , denoted by $o \sqsubseteq o^+$. Furthermore, if $o \neq o^+$, then $o \sqsubset o^+$, and we say o is properly contained in o^+ .
- Let $o_1, o_2 \in \mathcal{O}$, the union of o_1 and o_2 is a new object in \mathcal{O} , denoted by $o = o_1 \sqcup o_2$.

It is easy to see that \sqsubseteq is a partial ordering relation over the object set. This relation can be extended when we consider a more generic case where o^+ is produced with o as an input. Similarly, the operation \sqcup can be also extended by viewing o as the result produced with the inputs o_1 and o_2 . Considering that this paper focuses on security labels for data transferring, we restrict both \sqsubseteq and \sqcup to a simple case where o^+ is regarded as a new object obtained by adding extra information into o , and $o_1 \sqcup o_2$ is a new object generated by putting two objects together.

2.2 Security Classification and Mandatory Access Control (MAC)

Security classification, or level, is regarded as a hierarchical indicator of the degree of sensitivity to certain threats. For a given system, assume we have a particular set of security levels associated with a partial ordering relation. Formally,

Definition 1 We call (\mathcal{L}, \leq) a security classification system, where \mathcal{L} is a set of security levels, and \leq is a partial ordering relation defined on \mathcal{L} . For any $m, l \in \mathcal{L}$, $m \leq l$ means that l is a higher security level than m , and is read as “ m is dominated by l ”. Furthermore, if $m \leq l$ but $m \neq l$, then we write $m < l$ and say that m is strictly dominated by l .

Note that the infix notation $m \leq l$ means the same as $(m, l) \in \leq$. Therefore, $(m, l) \notin \leq$ means that m is not dominated by l .

Without loss of generality, the security classification system (\mathcal{L}, \leq) can be considered a (finite) lattice. In fact, every partial ordering set can be embedded in a lattice by including additional security levels if needed (Sandhu 1993). As a lattice, (\mathcal{L}, \leq) has two operators which are derived from the relation \leq . Formally, we have the definition as follows:

$$\begin{aligned} l \oplus m &= l.u.b.\{l, m\} \\ l \odot m &= g.l.b.\{l, m\} \end{aligned}$$

where $l.u.b.\{l, m\}$ is the least upper bound of the set $\{l, m\}$, that is, assume $l.u.b.\{l, m\} = a$, then $l \leq a$, $m \leq a$ and, for any b , if $l \leq b$ and $m \leq b$ then $a \leq b$. Symmetrically, $g.l.b.\{l, m\}$ is called the greatest lower bound of the set $\{l, m\}$. We will see in our discussion that the operation \oplus is the more useful of the two.

There also exists the unique element, say l_{min} , such that for all $l \in \mathcal{L}$, $l_{min} \leq l$, and, symmetrically, there exists the unique element, say l_{max} , such that for all $l \in \mathcal{L}$, $l \leq l_{max}$. Thus, l_{min} and l_{max} are called the minimum element and the maximum element of this lattice respectively or, accordingly, the lowest security level and the highest security level of the classification system.

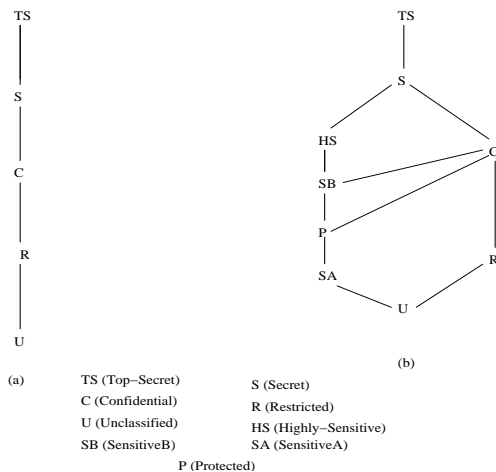


Figure 1: Security Classification Systems

Figure 1 illustrates two classification systems. The classification system in Figure 1(a) is a linear hierarchy, consisting of five security levels U (unclassified), R (restricted), C (confidential), S (secret) and TS (top-secret) with the ordering relation $U < R < C < S < TS$. This kind of linear security classification system, typically used in the military and government

sectors, can have any number of security classes. In the linear hierarchy, there are no incomparable security classes, and the definition of $l \oplus m$ and $l \odot m$ are then simply the maximum and the minimum, respectively, of l and m with respect to the relation \leq , e.g., we have $R \oplus S = S$ and $R \odot S = R$. In a more generic classification system, such as the one in Figure 1(b), there are some incomparable security classes, e.g., HS and C , but we have $HS \oplus C = S$, and $HS \odot C = SB$. In Figure 1, both of the systems have U as their lowest security level and TS as their highest. We can show that it is possible to implement a single security labelling system which encompasses multiple classification systems based on the lattice classification method.

Security policies may include inconsistencies, ambiguities, gaps, and conflicts with related policies. Usually, the reason for this is the lack of techniques for accurately expressing complex and detailed instructions written in a natural language. Therefore, it is important to provide a formal methodology that can be applied for expressing and reasoning about a variety of security policies.

In the following, we show that the mandatory access control policy can be expressed in terms of security classes attached to subjects and objects. Formally, there is a function that assigns a security level to each object and subject of a system:

$$\mu : \quad \mathcal{S} \cup \mathcal{O} \rightarrow \mathcal{L}$$

Any particular security level associated with an object implies a specific level of protection for the object, according to the enforced security policy, while a security level associated with a subject represents the level of clearance of the subject.

With the function μ above, the mandatory access policy, BLP rules (see (Sandhu 1993)), can be expressed as follows:

- *No-read-up rule*: Subject s can read object o if and only if $\mu(o) \leq \mu(s)$.
- *No-write-down rule*: Subject s can write object o only if $\mu(s) \leq \mu(o)$.

2.3 Categories and Need-to-Know Policy

Categories are a different concept from security levels. For example, categories may correspond to different tasks in a system. Agents with the same security level may not have the same right to access an object with a particular category. According to BLP rules, a subject with a certain security level is usually allowed to read a file (an object) with the same or lower security level. However, in some situations, such as within an implementation of a military project, some individuals may not have a need-to-know for this project, although they may have the same or higher security level than the one assigned to the project files. Therefore, there is a need to study the notation of categories deeply with the purpose of providing an appropriate access control model for such systems. We will show that combining categories together with security levels can rule out access to resources by individuals who do not have a need-to-know.

Given the object set \mathcal{O} , the subject set \mathcal{S} , the security classification system (\mathcal{L}, \leq) and a function μ , we define

$$\mathcal{O}_l(\mu) = \{o \in \mathcal{O} \mid \mu(o) = l\}, \text{ where } l \in \mathcal{L}.$$

That is, $\mathcal{O}_l(\mu)$ is defined as the set consisting of all objects whose security level is l , for simplicity, we write it as \mathcal{O}_l . Thus, for any object o , since it has exactly one security level, there is an unique set, i.e., $\mathcal{O}_{\mu(o)}$, such that $o \in \mathcal{O}_{\mu(o)}$. Therefore, for any $l, m \in \mathcal{L}$, if $l \neq m$, then $\mathcal{O}_l \cap \mathcal{O}_m = \emptyset$, the empty set. Furthermore, we have $\bigcup_{l \in \mathcal{L}} \mathcal{O}_l = \mathcal{O}$, which can easily be proved, so the sets \mathcal{O}_l partition \mathcal{O} .

Introducing (new) categories into a system may have a variety of reasons, such as new projects being established and restricted to be known by only some agents. Assuming there is a set of categories defined for a given system, then by defining an operation/relation on this set, we are able to form a category algebra for this system. Formally, we have:

Definition 2 A system, $(\mathcal{C}, |)$, is called a category algebra, where \mathcal{C} is the category set and $|$ is a binary operator defined on \mathcal{C} , that satisfy the following conditions:

- $\epsilon \in \mathcal{C}$, where ϵ is the null category;
- For any $c, e \in \mathcal{C}$, $c|e = e|c$. We may read $c|e$ as ' c or e ', which represents a new category produced by combining two categories as one with the operation $|$;
- For any $c \in \mathcal{C}$, $c|\epsilon = c$, and $c|c = c$.

According to this definition, a partial ordering relation \leq can be introduced with the following definition:

- For all $c \in \mathcal{C}$, $\epsilon \leq c$; and $c \leq c|e$ for any e in \mathcal{C} .

Furthermore, we say a category is *single* if it is not a combination of two or more actual categories with the operation $|$. An actual category means that it is not the null category ϵ . In contrast, a combination of two or more actual categories is not a single category.

Thus, the category algebra $(\mathcal{C}, |)$ is also expressed as the partial ordering set (\mathcal{C}, \leq) . We can then embed the partial ordering set in a lattice by including additional categories as required. For example, $\mathcal{C} = \{\epsilon, c, e\}$, we need to add $c|e$ to \mathcal{C} , (\mathcal{C}, \leq) then becomes a lattice. Therefore, in general, we can view the category algebra as a lattice. Note that the partial relation in (\mathcal{C}, \leq) is in fact a different relation from the one in (\mathcal{L}, \leq) , but without confusion, we denote both by \leq .

In the following, we discuss how to combine security levels with categories to establish an extended security classification system. Given a security classification system (\mathcal{L}, \leq) , and a category algebra (\mathcal{C}, \leq) , we first obtain a subset, say Ω , of $\mathcal{L} \times \mathcal{C}$ by the recursive definition given below.

- For all $l \in \mathcal{L}$, $(l, \epsilon) \in \Omega$,
- If an object with category c is contained or will be created in \mathcal{O}_l , we have $c \in \mathcal{C}$ and then $(l, c) \in \Omega$.
- For all $l, m \in \mathcal{L}$, if $l \leq m$ and $(l, c) \in \Omega$, then $(m, c) \in \Omega$.
- For all $l \in \mathcal{L}$, if $(l, c) \in \Omega$ and $(l, e) \in \Omega$, then $(l, c|e) \in \Omega$.

Then we define a partial ordering relation, \preceq , on Ω as follows:

- For any $(l, c) \in \Omega$ and $(m, e) \in \Omega$, $(l, c) \preceq (m, e)$ if and only if $l \leq m$ and $c \leq e$.

Thus, (Ω, \preceq) is an extended security classification system with categories. It is also a lattice classification system. Figure 2 is an example of such a classification system, where the set of security levels is $\{U, R, C, S, TS\}$ and the set of categories contains only three different elements (i.e. three single categories), X, Y and Z , not including ϵ .

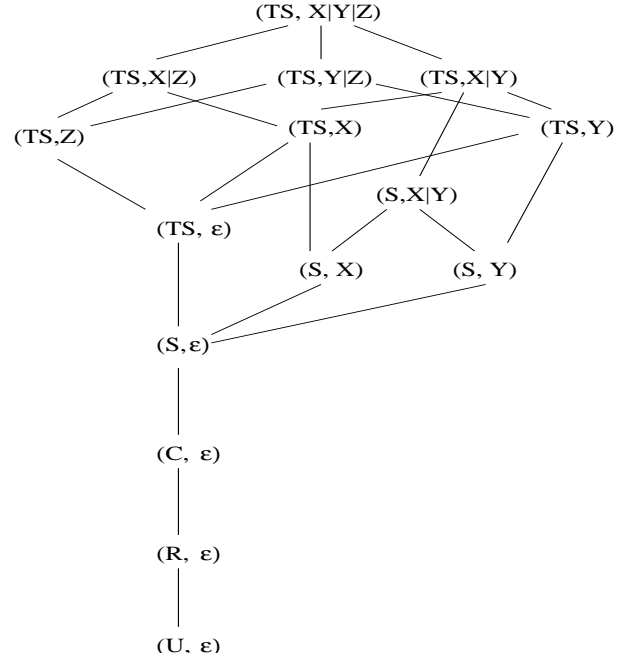


Figure 2: A Classification System with Categories

It is easy to show that, if $\mathcal{C} = \{\epsilon\}$, i.e., there is no actual category, then (Ω, \preceq) and the classification system (\mathcal{L}, \leq) are isomorphic.

A need-to-know security policy can be expressed in terms of the security levels and categories attached to subjects and objects. Formally, there is a function that assigns a security level with categories to each object or subject within a system:

$$\lambda : S \cup \mathcal{O} \rightarrow \Omega$$

$\lambda(o) = (l, c)$ implies the category of o is c and the specific level of protection for this object is l according to the security policy being enforced, while $\lambda(s) = (l, c)$ gives the clearance of the subject s . Note that, if there is no specific category assigned to an object o with the security level l , then $\lambda(o) = (l, \epsilon)$. Similarly for subjects.

Thus, with the function, a need-to-know policy can be expressed as:

- *Need-to-know rule*: Object o can be accessed by subject s if and only if $\lambda(o) \preceq \lambda(s)$.
- *Object-creating rule*: Subject s can write object o only if $\mu(s) = \mu(o)$ and $\lambda(o) \preceq \lambda(s)$.

The need-to-know rule is used to guarantee that there is no one who is able to access resources for which he does not have a need-to-know. The object-creating rule is similar to the no-write-down rule, which guarantees that any subject can only write objects with the same security level as the subject possesses.

3 Security Labelling Principles

In a system, objects are created and destroyed dynamically. There is no bound on the number of objects that can be created, but the assumption that at all times the set of objects is finite is appropriate, therefore, practically we are able to label all objects. We note that, when an object is created, a security level must be assigned to it at the same time by the applicable security policy. Also, with some applications, objects may need to be assigned an appropriate category as well.

Based on the given mandatory access policy (BLP rules) above, the following principles should be applied in assigning security levels to objects: For all $o, o_1, o_2 \in \mathcal{O}$, and any $s \in \mathcal{S}$,

- (1) If object o is created (owned) by the subject s , then $\mu(o) = \mu(s)$.
- (2) $\mu(o) \leq \mu(o^+)$, for any object o^+ in which o is contained.
- (3) $\mu(o_1 \sqcup o_2) = \mu(o_1) \oplus \mu(o_2)$.

These principles give a method (constraint) for assigning a security level to any (new) object. As we will see, the security level of an object must be contained, as a field, in the object's label. Therefore the principle actually provides a technique for generating (part of) the label.

It is easy to show that these principles are consistent with the mandatory access BLP rules. In other words, these principles satisfy the mandatory access policy. In fact, according to (1), the subject is not allowed to create an object whose security level is lower than the subject's security level (no-write-down); and, according to (2) and (3), subjects whose security level is lower than an object's level can never get the information contained in this object (no-read-up).

In some applications, principle 1 may be shifted as follows:

- (1') If object o is created (owned) by the subject s , then $\mu(s) \leq \mu(o)$.

However, with this principle, there must be some auxiliary rules for determining the exact security level that is assigned to any particular object generated by a subject. Similarly, in principle 2, $\mu(o^+)$ depends on what is added to o to obtain o^+ . Therefore, in a practical application, we may also need some auxiliary rules for determining $\mu(o^+)$, the security level of o^+ , case by case.

Based on a need-to-know policy, we have the following principles for assigning security levels and categories to objects: For all $o, o_1, o_2 \in \mathcal{O}$, and any $s \in \mathcal{S}$,

- (4) If object o is created (owned) by the subject s , then $\mu(o) = \mu(s)$ and $\lambda(o) \preceq \lambda(s)$.
- (5) $\lambda(o) \preceq \lambda(o^+)$.
- (6) Assume $\lambda(o_1) = (l_1, c_1)$ and $\lambda(o_2) = (l_2, c_2)$, then $\lambda(o_1 \sqcup o_2) = (l_1 \oplus l_2, c_1|c_2)$.

It is also easy to show that principles (4) – (6) are consistent with the need-to-know and object-creating rules that are employed to express the need-to-know policy.

4 A Dynamic Model for Security Labelling

In this section, we present a model for security labelling systems. Additionally, if a labelling system does not provide sound labels that satisfy the required security properties, then it cannot be regarded as a correct system. In particular, if the security policy within a label is not satisfied, then the label is useless. Therefore, we follow with a discussion on label validation, as well as some issues regarding relabelling within this model.

4.1 Security Tags and Label Format

A security label attached to an object should address the range of security aspects required by the applicable security policy. These include the level of protection to be given to the object, who is authorised to access the object, how it can be transmitted between systems, and so on. Information can be encoded in security labels by the use of security tags or tag sets. A *security tag* is an information unit that contains a representation of certain security-related information (e.g., a restrictive attribute bit map). A named *tag set* is the field containing a *TagSetName*, which may be a numeric identifier, and its associated set of security tags.

Caveats are a concept closely related to security tags. We assume that in a system/organisation there are a number of standard caveats from which the user (label creator) may choose when generating security labels. A caveat has the following format:

$$(TagType, TagName, -)$$

where “-” is a field associated with the *TagName* and may be used to contain caveat qualifiers. If no qualifiers are required, then this is deemed by the specification to have the value “NULL”.

There are five basic defined security tag types that are recommended for use in labelling systems (ITU-T Recommendation X.841 2000). They are *Restrictive bit map* (REST), *Enumerated* (ENUM), *Range* (RANG), *Permissive bit map* (PERM), and *Informative* (INFO).

TagName gives the name of the tag, such as, for example, *SendTo* and *DeptOnly*. The meaning of any *TagName* should be defined at the design stage of the labelling system development, for instance, *SendTo* stands for “send to” and *DeptOnly* for “department members only”. Two example caveats are given as follows:

$$\begin{aligned} w &= (PERM, SendTo, \{A_1, \dots, A_n\}) \\ u &= (REST, DeptOnly, -) \end{aligned}$$

A *security label* can consist of one or more security tag sets and, informally, it is a marking that is bound to an object, designating the security attributes of that object. In this view, a *security labelling system* is a system that is used to generate security labels for information objects that need to be labelled.

To simplify our discussion, we assume that each object will be assigned only one label at a time, and every label has only one tag set. In practice, a label may contain information regarding a security label identifier and the length of the label etc. In compliance with X.841 structures (ITU-T Recommendation X.841 2000), security labels are defined as having the following format:

$$[PolicyID, Classification, Category, TagSet]$$

where the field *PolicyID* is the object identifier for the applicable policy, the field *Classification* is the security level of the object to be labelled, the field *Category* is the security category of this object, and the field *TagSet* is a set of caveats.

As an example, given a security policy, p , and two caveats, w and u , as above, then we may have labels $[p, S, X, w]$, $[p, S, X, u]$, and $[p, S, X, \{w, u\}]$, where we assume S to be the security level and X the category shown in Figure 2.

Liu and Örgün (Liu & Örgün 2006) have pointed out that for any label with a single caveat, based on the definition of the caveat together with the security level and the category (as it appears in the label format proposed in this paper), one should be able to identify the subjects to whom an object with this label can be delivered or who are permitted to access this object. For example, an object labelled with $[p, S, X, w]$ can be delivered to only those agents who are in $\{A_1, \dots, A_n\}$ and hold a security clearance equal to or higher than S and have the right to access category X ; and an object labelled with $[p, S, X, u]$ can be delivered to only those department members who hold a security clearance equal to or higher than S and have the right to access category X . For a label containing more than one caveat, the delivering domain can be obtained by performing an intersection of the delivering domains associated with each individual caveat in the tag set. For example, an object labelled with $[p, S, X, \{w, u\}]$ can be delivered to only those who are both department members and in the set $\{A_1, \dots, A_n\}$, hold a security clearance equal to or higher than S , and have the right to access category X .

4.2 The Model

In the discussion of security labelling models, we propose a framework for security labelling as follows:

Definition 3 Given an open system G , let S be the subject set, \mathcal{O} the object set, \mathcal{P} the policy set, and (\mathcal{L}, \leq) the lattice-based security classification system and $(\mathcal{C}, |)$ the security category algebra. Let $\Omega \subseteq \mathcal{L} \times \mathcal{C}$ and \preceq be the partial ordering relation over Ω , derived from (\mathcal{L}, \leq) and $(\mathcal{C}, |)$ mentioned as above, such that (Ω, \preceq) forms a security classification system with categories for G ; and let λ be a total function defined from $S \cup \mathcal{O}$ to Ω :

$$\lambda : S \cup \mathcal{O} \rightarrow \Omega,$$

then we call $\langle S, \mathcal{O}, \mathcal{P}, (\Omega, \preceq), \lambda \rangle$ the security labelling framework for the system G .

The framework changes dynamically. The changes come from two aspects: dynamic changes to the security policies, which may directly lead to changes in (Ω, \preceq) , and dynamic changes to subjects and objects (the sets S and \mathcal{O}), from which corresponding changes to the function λ may result. Considering that security policies are relatively fixed for some period of time within a system, at any given time or within a reasonable time-period, the framework is definitely determined.

Based on the security labelling framework, a formal definition for security labelling models is given as follows:

Definition 4 Let $\Theta = \langle S, \mathcal{O}, \mathcal{P}, (\Omega, \preceq), \lambda \rangle$ be the security labelling framework for a given system G . Then a model for security labelling is a function depending

on the framework, denoted $\nu(\Theta)$, that assigns a label to each object. That is, for any object $o \in \mathcal{O}$, we have the security label $\nu(\Theta)(o) = [p, l, c, \{w, \dots, u\}]$ bound to the object, where $p \in \mathcal{P}$, $(l, c) \in \Omega$ and $\lambda(o) = (l, c)$, and w, \dots, u are taken from the set of caveats. We may simply call $\nu(\Theta)$ a security labelling system for the system G .

Intuitively, a security labelling system for a given system is a function based on the security labelling framework, which assigns labels to all objects in this system.

Since the function $\nu(\Theta)$ depends on the security labelling framework Θ and Θ can change dynamically, the model is a dynamic model. That is, at any particular time, the label assigned to each object depends on the security labelling framework at that time. This model is distinguished from the mechanically checkable model proposed in (Liu & Örgün 2006) in that the checkable model is, in fact, a static model that does not address the effects of dynamic changes to the framework and therefore does not support relabelling directly; while the model proposed in this paper captures the dynamic changes within a system, and provides support for relabelling when necessary.

4.3 Security Domains and Label Validation

As we said before, access control policies are the basis on which security labelling relies. An access control policy is defined not only in terms of its access control rules but also in terms of the access control domain that determines the subjects and objects to which the policy applies (Bidan & Issarny 1998). Informally, a security domain is a collection of subjects, to which a single security policy applies that is administered by a single authority.

Assume that an organisation has a security policy as follows:

- only those who have the key k , used to open a restricted room, can enter that room.

This policy may formally be expressed by

$$P : \forall x \forall r (\text{restricted}(r) \wedge \text{enter}(x, r) \rightarrow \text{has}(x, k_r)).$$

where k_r is the key to open the room r . Obviously, the security domain related to a specific policy is a subset of the subject set in a given system. With the policy P above, the security domain should consist of all individuals who are capable of walking from one place to another (including, for example, human beings, robots etc.); any other subjects do not belong to this domain. In other words, the security domain is a set of all subjects that are affected by this policy, while other subjects are not involved in this policy.

We now consider some notions related to security domains, which can be used for label validation.

Definition 5 Let p be a security policy, and o an object. A policy-implementation domain related to p and o can be expressed by

$$PID(p, o) = \{s \mid s \in SD(p) \ \& \ p^*(s, o)\},$$

where $SD(p)$ is the security domain for the policy p and p^* is regarded as a property corresponding to the policy p . That is, the domain $PID(p, o)$, as a subset of $SD(p)$, consists of all subjects that satisfy the policy p regarding the access to the object o .

With the policy P above, consider a restricted room A , denoted rA , then the property applied to find the policy-implementation domain could be $p^*: has(s, k_{rA})$. Thus, we have $PID(P, rA) = \{s | has(s, k_{rA})\}$, which consists of all individuals who can enter the room A .

The policy-implementation domain provides a mechanism (or an algorithm) for guaranteeing that the policy is satisfied. According to the above definition, the policy-implementation domain seems to depend on the security property that corresponds to the policy, therefore, we may have a number of different expressions for the domain, but the policy-implementation domain itself is unique.

With an access control policy, the policy-implementation domain explicitly gives the scope containing all individuals (subjects) who are allowed to access an object under the policy. The security label that is bound to an object provides the actual domain that contains those individuals who may finally be able to touch (or have) the object according to this label. Recall the example above, assume that the label $[P, R, \epsilon, (REST, DeptOnly, _)]$ is bound to rA . In the label, P is only the object identifier of the policy. We may not know whether other fields in the label are consistent with the policy, as the policy's actual content does not appear in the label, while the label implementation seems to be based only on other fields. Therefore, there is another notion, called a label-implementation domain, that may not be the same as the policy-implementation domain. We have:

Definition 6 Let α be the label bound to an object o . The label-implementation domain of the object o based on α , denoted $LID(\alpha, o)$, is defined by $LID(\alpha, o) = \{s | can_have(s, o) \text{ based on } \alpha\}$, where $can_have(s, o)$ means that the subject s may access o according to the label α .

Continuing the example of the restricted room, let $\alpha = [P, R, \epsilon, (REST, DeptOnly, _)]$ be the label bound to the room rA , then there are two cases:

- case 1: $LID(\alpha, rA) = PID(P, rA)$. The case holds if the key that is used to open the room rA is issued only to those who are in the department and have the security clearance equal to or higher than R .
- case 2: $LID(\alpha, rA) \neq PID(P, rA)$. This case happens when someone has the security clearance equal to or higher than R and is not in the department, but has the key to open the room rA , or someone who has no key to open this room but is in the department are allowed to enter the room.

Furthermore, we have:

Definition 7 Given the model $\nu(\Theta)$ and an object o , let $\nu(\Theta)(o) = \alpha = [p, l, c, \{w, \dots, u\}]$, then we say that the label α violates the policy p if $LID(p, o) \neq PID(p, o)$; and we say α is consistent with the policy p or α is valid if $LID(p, o) = PID(p, o)$.

4.4 Relabelling

Given a system G , assume that initially, say at time 0, the security labelling framework for the system is Θ and there is no change to Θ until time t , then at time t the security labelling framework for G remains Θ . More generally, we have:

Definition 8 Assume that at time t_1 the security labelling framework for a given system G is Θ and there is no change to Θ until time t_2 , then we say the security labelling framework for the system G is relatively static in the time interval $[t_1, t_2]$.

Thus, the following assertion is straightforward.

- Given a system G , assume that at time t_1 the security labelling framework for the system is Θ and it is relatively static in the time interval $[t_1, t_2]$, then if a label is valid at t_1 , then it is valid at all times in $[t_1, t_2]$.

That is, if the security labelling framework has not changed, then there is no need for relabelling.

Inversely, if at time t , the security labelling framework for a system is Θ but there are some changes to Θ , then it is obvious that at the next moment in time, $t + 1$, Θ is no longer the security labelling framework for the system.

There are several problems that arise from such dynamic changes to the security labelling framework which are worth investigating. They are:

- For any given object $o \in \mathcal{O}$, when does relabelling become necessary? In other words, is the label $\nu(\Theta)(o)$ no longer valid at time $t + 1$? Or do we have $\nu(\Theta')(o) = \nu(\Theta)(o)$ where Θ' is the security labelling framework at $t + 1$?
- If there is a need for relabelling, how do we deal with it?

Assume $\Theta = \langle \mathcal{S}, \mathcal{O}, \mathcal{P}, (\Omega, \preceq), \lambda \rangle$ and $\Theta' = \langle \mathcal{S}', \mathcal{O}', \mathcal{P}', (\Omega', \preceq'), \lambda' \rangle$ are the security labelling frameworks for the system at time t and at $t + 1$, respectively. Θ' is the framework that results from applying changes to Θ . Since changes can be complex, it is not easy to analyse the effects of a variety of changes to the security labelling framework. We do not intend to investigate all such changes here, but consider two simple cases as examples to provide some techniques for dealing with the relabelling issues.

Case 1: In this case, the only change is that several new objects are created at t and added to the object set. Thus, we have $\mathcal{S}' = \mathcal{S}$, $\mathcal{P}' = \mathcal{P}$, $\Omega' = \Omega$. The differences between Θ and Θ' are: $\mathcal{O} \subset \mathcal{O}'$ and $\lambda \neq \lambda'$. However, because the policy set and the classification system are not changed, the function λ' must satisfy that

$$\lambda'(x) = \begin{cases} \lambda(x) & \text{if } x \in \mathcal{S} \cup \mathcal{O} \\ (l, c) & \text{otherwise} \end{cases}$$

Therefore, in this case, for all $o \in \mathcal{O}'$, if $o \in \mathcal{O}$, then there is no need for relabelling; if $o \notin \mathcal{O}$, we assign a label to it.

Case 2: In this case, we assume the only change is that the policy p is replaced by p' . Then relabelling involves only those objects whose label contains the object identifier of the policy p in the the field *PolicyID*.

Assume $\Theta = \langle \mathcal{S}, \mathcal{O}, \mathcal{P}, (\Omega, \preceq), \lambda \rangle$ is currently the security labelling framework for a system, and there is a change σ to Θ , then at the next moment in time there must be some objects in \mathcal{O} that require relabelling. One question is whether we can, based on the change σ to Θ , determine the maximum subset

of \mathcal{O} where no object needs to be relabelled; or what kind of changes would allow one to determine such a maximum subset. The other question is how to define a relabelling function, such as the one suggested by Foley *et al* (Foley et al 1996), cooperatively within our model to satisfy the requirements for relabelling. We leave such problems for future work.

5 Implementation – Technical Issues

The implementation of our model involves many technical aspects, including the representation of the label syntax, the specification of security policy, label creation and validation, label binding mechanisms, etc. Implementing a practical security labelling system is not in the scope of this paper. Instead, we consider several technical problems, and identify some essential elements in implementing a security labelling system.

In this section, we first discuss the syntactical representation of labels and the technique for the specification of security policies, then explore two specific aspects – techniques for binding security labels to objects, and the basic components for implementing a security labelling system.

5.1 Label Specification

Having given the label format, the specification of labels can be done in many different ways. One way is based on the (traditional) formal specification technique, where the labels are specified with an abstract syntax, such as the specification proposed in (Liu & Orgun 2006). The advantage of this method is that such a specification provides a clear hierarchical structure for the system designer, and it is easily understood; however, the specification may be a little far from the actual programming language adopted in the implementation.

ASN.1 (Dubuisson 2000) also gives formal notations for high level specification of information to be exchanged between interacting applications, operating systems, programming languages, or other specifics associated with these applications. Considering that dynamic changes may often happen within a security labelling framework, the ASN.1 specification technique would be a good candidate for label specification, as recommended in (ITU-T Recommendation X.841 2000). With this notation, the security label is specified as follows:

```

Label ::= SEQUENCE {
    policyID      SecurityPolicyIdentifier
    classification SecurityLevel
    category      SecurityCategory
    tagSet        TagSet }

SecurityPolicyIdentifier ::= ObjectIdentifier
SecurityLevel ::= NaturalNumber
SecurityCategory ::= AlphabetLetter
TagSet ::= SET OF SecurityTag

SecurityTag ::= SEQUENCE {
    tagType      TagType
    tagName      TagName
    associateField AssociateField }

TagType ::= CHOICE of Type
TagName ::= ObjectIdentifier
AssociateField ::= PrivacyMark (Optional)

```

With the type `SecurityLevel`, there must be a function to map any security level of the set \mathcal{L} to a natural number representing the security level. As a result, only classifications that are total orders (such as Figure 1(a)) can be supported. Similarly with `SecurityCategory`, there needs to be a function to map any security category of the set \mathcal{C} to a letter in a given alphabet representing the category. The specification may also rely on the availability of a registration service to assign `TagSetName` and serve as the repository of the semantics, special handling rules, and other details required for the use of security policy-specific label sets. When security labels are associated with or require processing by a specific application, the `TagSet` element of the security labels can also be used to carry application specific data.

The ASN.1 format provides a clear structure for specifying security labels that meet the requirements of our proposed security labelling model. In addition, the ASN.1 specification can be easily mapped to actual programming code in the implementation stage.

5.2 Security Policy Information File

Security policy is the basis for decisions made by access control mechanisms. Domain-specific security policy is conveyed via the Security Policy Information File (ITU-T Recommendation X.841 2000). The Security Policy Information File (SPIF) contains a sequence of several items, including **securityClassifications**, which maps the classification in the security label to a classification in the clearance attribute, **securityCategories**, which maps the security categories in the security label to the security categories in the clearance attribute, and so on. The SPIF can also be defined with an ASN.1 specification.

Here we would only mention that an interpretation of a SPIF may be ambiguous, since the SPIF is usually obtained based on a security policy written in a natural language. Therefore, in the implementation of our model, it is important to provide a technique for translating the SPIF to a formal representation. Such formal representation of the SPIF not only allows agents to accurately interpret the policy, but also provides support for identifying whether there are conflicts between policies, and for proving the consistency of the label with the policy that is applied.

5.3 Binding Techniques for Labels

For binding the security label to an information object, we may or may not choose to use a cryptographic service. As a consequence, we have two methods for binding labels to objects stated as follows:

- *The method without the use of a cryptographic service.* In this method, a copy of the data, say object o , and a copy of the security label $\nu(\Theta)(o)$ are stored together, as a single data record, inside the secure boundary of the system. With this binding method, no cryptographic function is needed.
- *The method with the use of a cryptographic service.* In this method, a digital signature algorithm (*SigAlg*) together with the private key (K_s) for a public key algorithm is used. We can then generate the digital signature

$$\text{SigAlg}(K_s, H(o), \nu(\Theta)(o)),$$

for the object and its label, where H is a public function such that $H(o)$ does not reveal information about o . In this case, the digital signature $SigAlg(K_s, H(o), \nu(\Theta)(o))$ is stored together with the object o and the label $\nu(\Theta)(o)$ in a data record; the generated digital signature binds $\nu(\Theta)(o)$ to o . With this binding method, $\nu(\Theta)(o)$ and $SigAlg(K_s, H(o), \nu(\Theta)(o))$ need not be stored inside the secure boundary of the system.

With the first method, the system needs to be capable of protecting the integrity of the security label and the integrity, and possibly also the secrecy, of the data. The second method allows the use of the public key of the public key algorithm as a verification key to verify the signature. If a cryptographic service is invoked with an incorrect value of $\nu(\Theta)(o)$, o or $SigAlg(K_s, H(o), \nu(\Theta)(o))$, then the inconsistency is detected.

5.4 Main Components of a Labelling System

Intuitively, within a security labelling system, the labelling function accepts an object as input, and outputs a security label, which will be bound to this object. In order to implement the system, one needs to build a number of mechanisms to perform a variety of functions. In our model, the security labelling function is performed by three major mechanisms. Most of the mechanisms/components given below are similar to those that have been proposed in (Liu & Orgun 2006), but there is one new component, the RLM (Relabelling Mechanism), which is particularly related to the dynamic model and dealing with changes of environment and relabelling requirements.

- PIM (Policy Identification Mechanism): a mechanism to determine which policy is applied to a specific object;
- SCFM (Security-level & Category Finding Mechanism): a mechanism to find the security level and category for any given object to be labelled;
- CCM (Caveat Choosing Mechanism): a mechanism to determine what caveats may or should be chosen for constructing the label for an object;

All these mechanisms operate based on the security labelling framework, which consists of three modules, SOD (Subjects & Objects Database), PM (Policy Management), and TR (Tags Registration).

Other important components contained in our labelling system are:

- VRE (Validation & Reasoning Engine): a reasoning engine used to check the consistency of security labels produced. It directly connects with the labelling framework as well as the security labelling function.
- LBM (Label Binding Mechanism): a mechanism applied to bind a valid label to an object.
- RLM (Relabelling Mechanism): a mechanism dealing with dynamic changes to the security labelling framework and relabelling those objects that are affected by the changes.

Note that in most systems PIM, SLFM, and CCM require human input and judgement.

6 Other Issues

An access control model that supports the dynamic aspects of security labelling is becoming increasingly important in order to accurately represent the use of classifications and categories in the real world. While security classifications tend to remain relatively static, need-to-know categories and the associated policy need to be much more flexible. The access control model we have described addresses a number of issues associated with dynamics, including the dynamic creation and deletion of objects, dynamic creation and modification of policies and the relabelling of existing objects. To support this model, the label format needs to be extensible to allow for the creation of new categories and policies.

The model deals with dynamic changes by viewing the framework over time as a series of distinct static frameworks. This allows for changes without requiring the explicit notion of time. However, there are some limitations to this approach. In particular, a category that changes meaning over time will, in the described model, always require the creation of new categories to represent the different meanings. Additionally, objects associated with the original category may need to be relabelled and the policy changed to support the new framework. In some cases, this may be the most appropriate way of dealing with the issue. However, in other cases, the changes may be subtle or directly related to time, resulting in a proliferation of new categories and relabelling requirements in order to deal specifically with time-related issues. The explicit use of time in the model could alleviate this problem by allowing time-based constraints to be expressed in the policy, rather than in the categories themselves.

We also would mention an important but frequently neglected aspect of access control that relates to assurance requirements. There are several major aspects, but we refer to just two of them here.

Firstly, not all access control need necessarily be carried out to the same assurance level. Within Defence, the access controls relating to mandatory access control, particularly with respect to differences of classification, must be carried out at high levels of assurance. In contrast, many need-to-know restrictions may be implemented using very standard lower-assurance solutions, similar to those acceptable in commercial environments. These differences in assurance requirements are often captured in an ad hoc way. For example, the network encryption policy used to enforce MAC may be very rigorous, while the internal access control policy, used for need-to-know controls, may be less stringent. It would be preferable if these differences could be captured within a single high-level policy; we are investigating one such approach.

The other aspect relates to the assurance techniques involved in carrying out labelling. Currently most solutions aim to provide good trust at the release point, effectively the access control enforcement function. In some solutions, assurance is provided for the integrity of the data used in the access control decision, via the use of binding mechanisms, as discussed above, and in credential certificates for users. But there are other places within a security labelling framework where trust is required, most notably at the actual labelling point itself. Lack of assurance at this point is what results in the lower assurance of DAC over MAC, as discussed by Spalka et al (2000), but is equally problematic for an untrusted labelling system that can be exploited by Trojan processes.

7 Conclusion

Need-to-know policies are the basis for security labelling, and should be followed in the labelling process. This paper has proposed a lattice-based access control model for expressing such policies. The lattice-based model is not new, but we provide a formal technique for describing the kind of models that allow one to deal effectively with both security levels and categories within the security labelling process.

In this paper, we have also proposed a dynamic model for security labelling that supports the use of security categories with security classifications, as well as providing support for relabelling and label validation. This model provides a functional base for the design and implementation of a security labelling system.

Future work may include an extended discussion on the dynamic management of security policies and its implementation. The aim would be to discuss the combination of security labels and associated security mechanisms to achieve the required security goals. A further discussion on dynamic aspects and the implementation of assurance requirements associated with our model is needed; and a deep analysis of threats and attacks that directly affect security labels and labelling systems may also need to be considered.

References

- Bell, D. E. & LaPadula, L. J. (1976), Secure Computer System: Unified exposition and Multics interpretation. MTR-2997, MITRE, Bedford, MA.
- Biba, K. J. (1977), Integrity Consideration for Secure Computer Systems. MTR-3153, The Mitre Corporation.
- Bidan, C. & Issarny, V. (1998), Dealing with multi-policy security in large open distributed systems. In *Proceedings of 5th European Symposium on Research in Computer Security*, pages 51–66.
- Dubuisson, O. (2000), ASN.1 Communication Between Heterogeneous Systems. Translated from French by Philippe Fouquart. Available from <http://www.oss.com/asn1/booksintro.html>.
- Ferraiolo, D. F., Kuhn, D. R. & Chandramouli, R. (2003), Role-Based Access Control. Artech House Inc.
- FIPS PUB 188. (1994), *Standard Security Label for Information Transfer*. Available from www.itl.nist.gov/fipspubs/fip188.htm.
- Foley, S., Li, G. & Qian, X. (1996), A Security Model of Dynamic Labeling Providing a Tiered Approach to Verification. In *the IEEE Symposium on Security and Privacy*, pages 142–154, May 1996.
- Heintze, N. & Riecke, J. G. (1998), The slam calculus: Programming with secrecy and integrity. In *Proceedings of 25th ACM Symposium on Principles of Programming Languages (POPL)*, pages 365–377, San Diego, California.
- Housley, R. (1993), *Security Labeling Framework for the Internet*. Internet RFC 1457, May 1993.
- Internet CIPSO Working Group. (1993), *Common IP Security Option Version 2.3*. Internet Draft.
- ITU-T Recommendation X.841 (2000), *Information technology - Security Techniques - Security information objects for access control*.
- Liu, C. & Orgun, M. A. (2006), Towards security labelling. In V. Estivill-Castro and Gill Dobbie, editors, *Proceedings of the 29th Australasian Computer Science Conference, ACSC2006*, pages 69–76, Hobart, Australia. Australian Computer Society, Inc.
- Myers, A. C. & Liskov, B. (2000), Protecting privacy using the decentralized label model. *ACM Transactions on Software Engineering and Methodology*, 9(4):410–442.
- Sandhu, R. V. Lattice-based access control models. *IEEE Computer*, 26(11):9–19 (1993).
- Spalko, A., Cremers, A. B. & Lehmle, H. (2000), Protecting Confidentiality against Trojan Horse Programs in Discretionary Access System. In *Proc. Australasian Conference on Information Security and Privacy, ACISP 2000*, Brisbane, Australia, LNCS 1841:1–17, Springer.
- Weissman, C. (1969), Security control in the ADEPT-50 time-sharing system. In *AFIPS Conference Proceedings*, Vol. 35, pages 119–133. FJCC, 1969.
- Zdancewic, S. Zheng, L., Nystrom, N. & Myers, A.C. (2001), Untrusted hosts and confidentiality: Secure program partitioning. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, pages 1–14, Banff, Canada.
- Zheng, L. & Myers, A.C. (2004), Dynamic security labels and noninterference. In *Proceedings of the 2nd International Workshop on Formal Aspects in Security and Trust (FAST)*, Toulouse, France.

Cost-based and Time-based Analysis of DoS-resistance in HIP

Suratose Tritilanunt

Colin Boyd

Ernest Foo

Juan Manuel González Nieto

Information Security Institute
 Queensland University of Technology,
 GPO Box 2434, Brisbane, QLD 4001, Australia,
 Email: s.tritilanunt@student.qut.edu.au,
 {c.boyd, e.foo, j.gonzaleznieto}@qut.edu.au

Abstract

We develop a formal model of the Host Identity Protocol (HIP) based on Timed Coloured Petri Nets (Timed CPNs) and use a simulation approach provided in CPN Tools to achieve a formal analysis. We aim to examine unbalanced computation that leads to resource exhaustion attacks in key exchange protocols comparing among a legitimate initiator, four types of adversary who attempt to deny the service at different stages of the protocol execution, and a responder. By adopting the key idea of Meadows' cost-based framework and refining the definition of operational costs during the protocol execution, our simulation provides an accurate cost estimate of protocol execution comparing between those principals. Under four defined attack strategies, however, Meadows' cost-based framework generates a different outcome compared with the simulation approach from Timed CPNs. Analysis of our experimental results reveals a limitation of Meadows' cost-based framework for addressing DoS threats.

Keywords : Host Identity Protocol (HIP), Cost-based Framework, Timed Coloured Petri Nets

1 Introduction

Denial of service (DoS) attacks are serious threats to computer networks since they attempt to prevent the responder and legitimate users to establish connections. DoS attacks might attempt to overwhelm either network bandwidth or responder's resources, including memory and central processor unit (CPU). Although both of these targets are important, in this paper we focus on how to protect the responder from exhausting resources based on vulnerabilities of the underlying key exchange protocol.

Most key exchange protocols aim to establish and exchange cryptographic parameters. These protocols require the associated responder to spend some expensive computations, such as modular exponentiation in the case of Diffie-Hellman and RSA digital signature algorithm, while the initiator requires computations which are cheaper than for the responder. As a result, these protocols are susceptible to denial-of-service (DoS) attacks because adversaries can launch DoS attacks to overwhelm resources of the responder before the responder can detect the attack and authenticate identity of the launchers.

In order to design cryptographic protocols, researchers have developed general-purpose verification tools to use in modeling and verifying them over many years. One example formal method is Petri Nets. Although Petri Nets have been developed since 1962 by Petri (1962), they have been recently used to verify cryptographic and security protocols. There are two forms of Petri Nets: ordinary Petri Nets and high level Petri Nets. In this paper, we focus on high level Petri Nets, i.e. Colored Petri Nets, because they have several benefits in the analysis and verification of cryptographic protocols as stated by Jensen (1998b).

To develop a formal framework of cryptographic protocols, we use CPN Tools (2004) which is an interaction tool for modeling and analysing Coloured Petri Nets. Our formal model is developed based on Timed Coloured Petri Nets (Timed CPNs) and uses a simulation approach provided in CPN Tools to achieve a formal analysis. The goals of simulation of this protocol are to explore vulnerabilities of DoS-resistant protocols by examining unbalanced computation that leads to resource exhaustion attacks in key exchange protocols. Simulation approaches are valued in the research community not only for exploring vulnerabilities in cryptographic protocols, but also guaranteeing security services of such protocols similar to a mathematical approach. Using simulation approaches has several benefits over mathematical analysis; for instance, simulation provides *flexibility* to the developer to choose, examine and adjust parameters for evaluating the system. In addition, simulation provides *visualization* to users who can see and learn what is happening during the simulation of cryptographic protocols to gain more understanding for evaluating the correctness of those protocols.

In this work, we adopt the key idea of Meadows' cost-based framework (2001) and refine the definition of operational costs by assigning an accurate estimate of various cryptographic operations during the protocol execution. Meadows introduced a systematic framework to analyse DoS-resistant protocols by computing and comparing the cost incurred by both parties at each step in a key exchange protocol. Meadows analysed the STS protocol (a protocol without special DoS resistance properties) and later Ramachandran (2002) and Smith (2006) used Meadows' framework to analyse the JFK protocol (2002) in order to demonstrate its DoS prevention capabilities.

Surprisingly, there has been little interest in the research community in applying Meadows' framework to different protocols. Moreover, the limited application so far has suffered from two significant shortcomings which make the results of restricted value.

1. The cost analysis has only taken into account *honest* runs of the protocol. In principle, the adversary (typically the client in a client-server protocol) can deviate arbitrarily from the protocol in order to achieve an attack. By only taking

into account honest behaviour it is quite likely that logical attacks will be missed. While Meadows certainly recognised this fact, no research has yet examined the effectiveness of the framework in detecting such potential attacks.

2. Meadows used only a coarse measure of computational cost, with three levels denoted as cheap, medium or expensive. In practice it can be quite difficult to classify and compare operations in such a limited way. For example, in Meadows' classification digital signature generation and verification are counted as of equal cost, yet in practice an RSA signature generation may take 2 or 3 orders of magnitude more effort than RSA signature verification.

Motivated by the above two limitations, this paper provides a refinement of Meadows' cost-based framework. For our sample protocol we use the Host Identity Protocol (HIP) (Moskowitz 2006), which has built-in DoS resistance. Using coloured Petri nets as our formalism, we provide a formal specification of HIP to allow automatic searching of adversary and victim cost under four different adversarial attack strategies. Moreover, we examine the tolerance of HIP under different levels of puzzle difficulty as well as investigate the most effective attack strategies by measuring the successful throughput in order to compare the result with the prediction from Meadows' framework. Although our range of adversarial actions is limited, we believe that we have demonstrated that formal specification and analysis is an effective way to extend the scope and value of the Meadows' framework which paves the way for more detailed application of the framework.

The main contributions of this paper are:

- a refinement of Meadows' cost-based framework to more accurately represent the cost of typical cryptographic algorithms;
- the first formal specification and automatic analysis of Meadows' framework;
- simulation and analysis of HIP under normal conditions and four scenarios of DoS attacks;
- a time-based analysis that reveals a limitation of Meadows' cost-based framework for addressing DoS threats.

2 Background and Previous Work

The purpose of this section is to provide the background on the Meadows' cost-based framework and HIP, as well as the previous work on the analysis of security protocols using Coloured Petri Nets.

2.1 Meadows' Cost-Based Framework

Meadows' framework (2001, 2003) works by comparing cost to the attacker and cost to the defender, defined using a *cost set*¹. To model the protocol framework, we need to calculate the cost of the sequence of protocol actions, comparing between an attacker and a defender. However, each action could use different CPU or memory resources during a protocol execution. Therefore, we need to find an alternative way to calculate the precise total cost of these actions and the technique for comparing them.

Considering the characteristic of DoS attacks, there are two possible ways mentioned by Meadows (2001) to cause the defender to waste resources. First,

the defender may process a bogus instance of a message inserted by the attacker into a protocol. The cost to an attacker is the cost of creating and inserting the bogus message, while the cost to the defender is the cost of processing the bogus message until an attack is detected. Second, the defender participates in a bogus instance of the protocol with the attacker. The cost of this situation is equivalent to the cost of running the entire protocol until the defender can detect the attack or the attack stops.

At this stage, we limit the abilities of adversaries during the protocol execution to take one of a small number of possible actions when the protocol specifies that a message should be sent: adversaries either continue normally with the protocol or partially complete the protocol. Intuitively these are the most obvious ways for adversaries to make the defender use unwanted resources. In our examples adversaries send messages at two points in the protocol, so we limit adversaries to either attack the first message by flooding a large number of random messages to overwhelm the resources of the responder, or to attack the second message by faking its packets to waste the responder resources for verifying it.

2.2 Host Identity Protocol

The host identity protocol (HIP) has been developed by Moskowitz (2006). Later, Aura et al. (2005) found some vulnerabilities and proposed guidelines to strengthen the security of HIP. The base exchange of HIP is illustrated in Figure 1.

HIP is a four-packet exchange protocol which allows the initiator I on the IP address IP_I and responder R on the IP address IP_R to establish an authenticated communication. Both I and R hold long-term keys to generate signatures $Sig_I(\cdot)$ and $Sig_R(\cdot)$ respectively. It is assumed that both principals know the public key PK_I of the initiator and PK_R of the responder represented in the form of host identifiers (HI) in advance. HIT represents the host identity tag created by taking a cryptographic hash over HI .

A one-way hash function $H(\cdot)$ is used to form the puzzle, while H_{K_s} represents a keyed hash function using session key K_s to generate a hash-MAC (HMAC). The value s is a periodically changing secret only known to the responder. $LSB(t, k)$ takes as input a string t and a parameter k and returns the k least significant bits of t . 0^k is a string consisting of k zero bits. $E_{K_e}(\cdot)$ and $D_{K_e}(\cdot)$ denotes a symmetric encryption and decryption respectively under session key K_e . To generate session keys K_e and K_s , HIP employs Diffie-Hellman key agreement protocol. Parameters used to generate these keys consist of large prime numbers p and q , a generator g , a responder's secret value r , and an initiator's secret value i .

HIP adopts a proof-of-work scheme proposed by Jakobsson and Juels (1999) for countering resource exhaustion attacks. In a proof-of-work, HIP extends the concept of a *client puzzle*, first proposed by Juels and Brainard (1999), and later implemented by Aura et al. (2000) for protecting the responder against DoS attacks in authentication protocols. Moreover, HIP allows the additional feature of the client puzzle that helps the responder to delay state creation (Aura & Nikander 1997) until the checking of the second incoming message and the authentication has been performed in order to protect the responder against resource exhaustion attack.

2.3 Coloured Petri Nets (CPNs)

CPNs (Jensen 1998a, Jensen 1997) are one type of high-level nets based on the concept developed back

¹More details are in Appendix A.

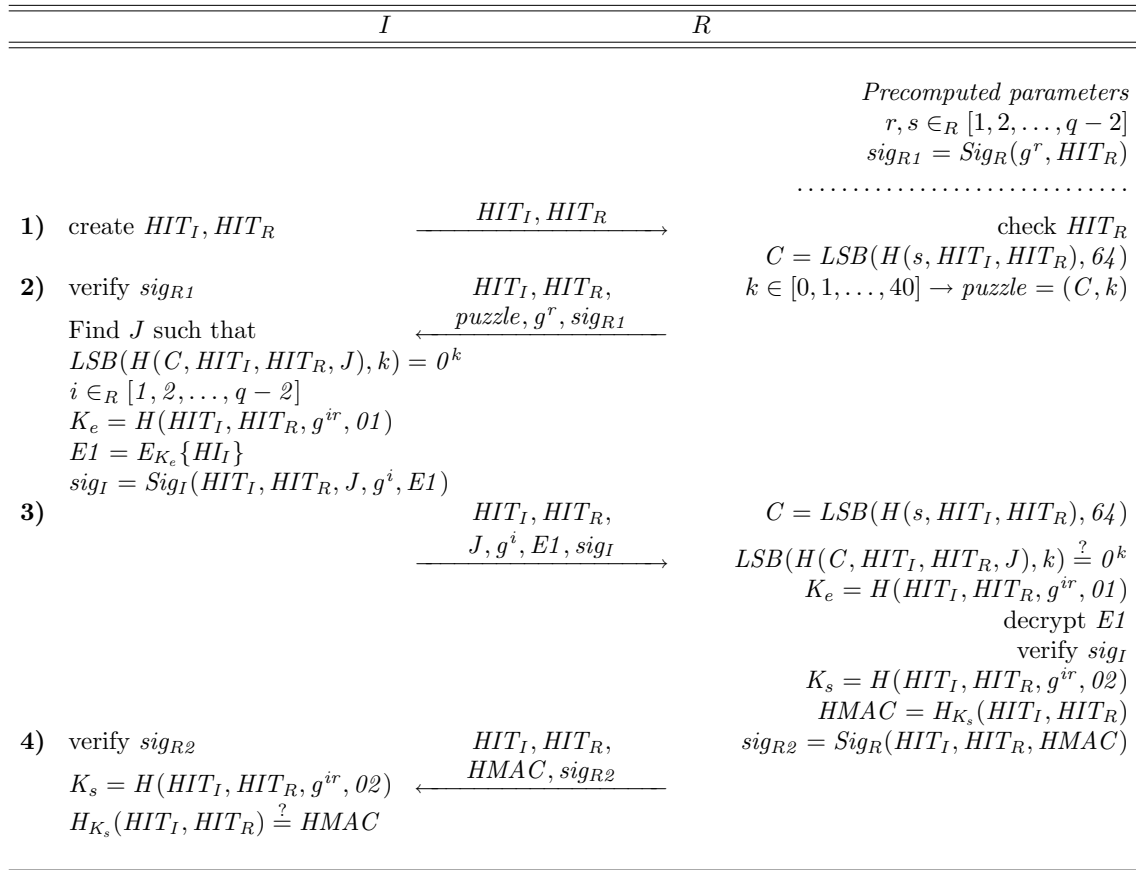


Figure 1: HIP Protocol

in 1962 by Petri (1962). CPNs is a state and action oriented model including places, transitions, and arcs.

Over many years, cryptographic and security protocols have been modeled and verified using CPNs. Neih and Tavares (1993) implemented models of cryptographic protocols in the form of Petri Nets. To explore vulnerabilities of such protocols, they allowed an implicit adversary with limited abilities to launch attacks and then examined the protocol using exhaustive forward execution. Doyle (1996) developed a model of three-pass mutual authentication and allowed an adversary to launch multiple iteration and parallel session attacks. Han (1996) adopted CPNs for constructing a reachability graph to insecure states and examining the final states in OAKLEY. Al-Azzoni (2004) has developed a model of Needham-Schroeder public key authentication protocol and Tatebayashi-Matsuzaki-Neuman (TMN) key exchange protocol.

To the best of our knowledge, there is no implementation of CPNs focusing on an exploration of unbalanced computational threats that lead to resource exhaustion attacks in key exchange protocols.

3 Cost-based Framework

This section provides a cryptographic benchmark of some specific algorithms which is an alternative technique to measure CPU usage for representing more specific costs instead of an original representation by Meadows. Finally, we present HIP by means of a cost-based specification.

3.1 Refinement of Meadows' Framework

An obvious limitation of the original formulation of the framework is that the computational costs are not

defined precisely, but consist instead of a small number of discrete values. Indeed Meadows (2001) herself called this a “crude and ad hoc cost function”. In order to obtain a more useful cost comparison we need to obtain a more accurate estimate of the computational and/or storage costs required to complete the protocol steps. How to do this is not as obvious as it may seem at first.

When comparing efficiency of different cryptographic protocols it is customary to count the number of different types of cryptographic operations. For protocols that use public key operations it is common to ignore most operations and count only the most expensive ones, which typically are exponentiations in different groups (traditionally written as multiplications in elliptic curve computations). However, for the protocols that interest us this is definitely not acceptable. As mentioned above, one common technique in DoS prevention is to demand that clients solve *puzzles* which require the client to engage in some computational effort, such as to iterate a hash function a large number of times. Although one hash computation takes minimal time in comparison with a public key operation, ignoring a large number of hash computations could make the cost function ignore completely the DoS prevention mechanism when a puzzle is used. Therefore we need to be able to compare directly the cost of all different kinds of cryptographic operations.

Comparing operations like hashing and exponentiations directly seems very hard to do since they are based on completely different types of primitive instructions. Therefore we have resorted to an empirical comparison which compares benchmark implementation on common types of processors. While we acknowledge that the detailed results may differ considerably for different computing environments (CPU,

Table 1: Computational Cost of CPU and Time Usage of Specific Algorithms

Hash	kCycle/Block	nsec/bit	Symmetrical Crypto	kCycle/Block	nsec/bit
SHA-1 (512bits/block)	1.89	1.84	DES (64bits/block)	0.75	5.86
HMAC/MD5 (512bits/block)	0.59	0.58	AES (128bits/block)	0.53	2.05
Public-Key Crypto	kCycle/ops	nsec/bit	Key Exchange	kCycle/ops	nsec/bit
RSA Encryption / RSA Verification	383.66	187.08	Diffie-Hellman Key-Pair Generation	4605.65	2245.80
RSA Decryption / RSA Signature	9985.47	4869.11	Diffie-Hellman Key Agreement	8100.69	3950.05

memory, and so on) we believe that the obtained figures are indicative of the true cost in typical environments and allow reasonable comparisons to be made.

For our cost estimates, we use the cryptographic protocol benchmarks of Wei Dai (2004). These include tested speed benchmarks for some of the most commonly used cryptographic algorithms using Crypto++ library² version 5.2.1 on a Pentium 4 2.1 GHz processor under Windows XP SP 1. More cryptographic benchmarking has been done by Gupta (2002) and Tan (2004) on the specific processors; however, they did not test public-key encryption.

Table 1 only presents the results for some specific cryptographic algorithms available for negotiating during the HIP based exchange defined in HIP specification. The units that we use in Table 1 are kilocycles per block (note that block size varies for different algorithms). This allows direct comparison of CPU usage and may be expected to be similar on processors with different clock speeds. This entails conversion from the original data which uses direct time measurements.

From the table, we are able to estimate the CPU usage in cycles per block for common hash functions and the symmetric key encryption, and cycles per operations for the 1024-bit key lengths of public-key encryption and Diffie-Hellman key exchange algorithm. Once we get a result, we scale it down by a factor of 1000 (kilo) and apply these costs in our formal specification and analysis. Before we can export these values into CPN Tools, we round them into an integer representation because CPN Tools limits only integers in the simulation process.

3.2 HIP Cost-Based Specification

We shall use HIP as an example in order to provide an analysis using Meadows' cost-based framework. By employing the concepts and definition from Appendix A, we are able to interpret HIP based exchange between two principals by means of cost-based specification as shown in Figure 2.

For the first message the initiator (I) performs $create_HIT_I, create_HIT_R$ and sends a requested message to the responder (R). Meanwhile, R performs the actions $pre_create_r, pre_create_s, pre_exp_g^r, pre_sign_sig_{R1}$ before a requested message arrives to the connection queue. These *pre-actions* are pre-computed in order to protect R from wasting some amount of resources at the initial state. Once R receives this request, it

spends only a cheap operation to check HIT_R for completing the first message. Considering a second message, R performs $compute_hash_C, create_k$ and sends $HIT_I, HIT_R, puzzle, sig_{R1}$ to I . When I receives this message, I then performs $verifysig_sig_{R1}, accept_1$. If $verifysig_sig_{R1}$ fails, the process stops. Otherwise, I accepts the message and then continues to brute-force search a puzzle solution $compute_hashsolution_j$.

To construct the third message, I performs $computekey_K_e, encrypt_{HI}, sign_sig_I$ appended with the signature sig_I and sends it to R . Having received the message from I , R begins to check C by recovering the message $R1$ counter and then hashing received HIT_I, HIT_R with the server secret s . If this process is verified correctly, R validates the puzzle solution by performing $verify_solution$. If invalid, the process stops. Otherwise, R performs $computekey_K_e, decrypt_{E1}$ for recovering HI_I . After that, R finally verifies the signature of I by performing $verifysig_sig_I$. Also, the output will be either success which causes the responder to continue to the next steps, or failure which causes the responder to reject and clear the connection queue. In the fourth message, R performs $computekey_K_s, compute_hash_{MAC}, sign_sig_{R2}$ and sends the message $HMAC, Sig_R[HIT_I, HIT_R, H_{K_s}(MAC)]$ to I . Finally, I begins to compute $computekey_K_s, verify_hash_{MAC}, verifysig_sig_{R2}, accept_3$ for key confirmation.

By using the original cost functions, the tolerance relation for the first message would be the pair (Cheap, Cheap) which is an acceptable result, while for the third message would include the pair (Medium, Cheap), or (Medium, Medium), or (Expensive, Expensive) which is also a reasonable result. However, these results are too coarse for achieving formal analysis of the computational costs. As a result, we replace such original cost measures suitable for hand analysis with more precise values from Table 1 in our simulation.

4 Automated Simulation by CPN Tools

This section provide the detail of the HIP cost-based and time-based model as well as the experimental results of these constructions.

4.1 The Construction of HIP in CPNs

HIP Cost-based Construction: In this model, we insert a special place to every cryptographic transition called a *cost-place* for capturing and displaying the computational cost of individual steps. This

²Available at <http://www.eskimo.com/~weidai/cryptlib.html> and <http://sourceforge.net/projects/cryptopp/>

MSG	Cost-based Function
1)	$I \rightarrow R :$ $create_HIT_I, create_HIT_R \parallel$ $HIT_I, HIT_R \parallel$ $pre_create_r, pre_create_s, pre_exp_g^r, pre_sign_sig_{R1}, verify_HIT_R$
2)	$R \rightarrow I :$ $compute_hash_C, create_k \parallel$ $HIT_I, HIT_R, puzzle, sig_{R1} \parallel$ $verifysig_sig_{R1}, accept_1$
3)	$I \rightarrow R :$ $compute_hashsolution_J, computekey_K_e, encrypt_{HI_I}, sign_sig_I \parallel$ $HIT_I, HIT_R, J, g^i, K_e\{HI_I\}, Sig_I[HIT_I, HIT_R, J, g^i, K_e\{HI_I\}] \parallel$ $verify_C, verify_solution_J, computekey_K_e, decrypt_{E1}, verifysig_sig_I, accept_2$
4)	$R \rightarrow I :$ $computekey_K_s, compute_hash_{MAC}, sign_sig_{R2} \parallel$ $HIT_I, HIT_R, HMAC, Sig_R[HIT_I, HIT_R, HMAC] \parallel$ $verifysig_sig_{R2}, computekey_K_s, verify_hash_{MAC}, accept_3$

Figure 2: HIP Protocol in the Cost-Based Framework Notation

process is performed by adding the CPU usage data from Table 1. Measurement of CPU usage indicates individual steps as well as the total cost compared between an initiator, every single type of adversary, and a responder, when such principals complete a single round of the protocol simulation.

HIP Time-based Construction: We attempt to design a more realistic model by adopting the concept of time into the HIP time-based simulation and analysis. That means every cryptographic process requires some amount of time calculated by using cryptographic benchmarks of Crypto++ library (Dai 2004). In the Timed CPNs, the concept of the *simulated time* or the *model time*³, which is represented by the system clock in the tool, has been introduced. Once we have attached the system time into tokens, we can see the sequence or action of states that tokens move to as a temporal analysis. It means only tokens which hold the current time as presented on the clock can be traversed to the next position, while the others have to wait until the system clock reaches their value.

Furthermore, the concept of the responder's resource is used in this evaluation. This resource represents the number of available connections in the queue to process the incoming requests. Once the responder has to deal with requests, the responder spends one unit of resource for each individual request. It means that if incoming packets exceed the responder capacity, the responder then rejects further incoming packets until he has either processed the legitimate traffic or detected bogus messages and removed them from the storage. Note that the process to order incoming packets arriving to the connection queue is non-deterministic because these packets have been arranged randomly by CPN Tools.

4.2 Adversary's Ability

Apart from the honest client (hc) who initiates the legitimate traffic, we allow four types of adversary who have the similar goal to deny the service of the responder by overwhelming the responder's resource.

Type 1 adversary (ad1) computes a valid first message (may be pre-computed in practice), and then takes no further action in the protocol.

Type 2 adversary (ad2) completes the protocol normally until the third message is sent and takes no further action after this. The computations of this adversary include searching for a correct client puzzle solution J , generating a session key K_e and encrypting a public key PK_I , and finally computing a digital signature Sig_I .

Type 3 adversary (ad3) completes the protocol step one and two with the exception that the adversary does not verify the responder signature sig_{R1} . The adversary searches for a correct client puzzle solution J but randomly chooses the remaining message elements: an encrypted element $K_e\{HI_I\}$ and a digital signature sig_I . The adversary takes no further action in the protocol.

Type 4 adversary (ad4) is like an adversary type 3, except that the client puzzle solution J is now also chosen randomly.

To clarify the description of adversaries' ability, the major goal of Type 1 adversary is to overwhelm the responder's storage by sending a large number of requested packets, for example, a denial-of-service attack via ping (CERT 1996a) and SYN flooding attack (CERT 1996b), while the major goal of Type 2, 3, and 4 adversary is to force the responder to waste computational resources up to the final step of the digital signature verification and digital signature generation which are expensive operations.

4.3 Experiment

To obtain experimental results, we set up two main different simulations;

Experiment 1: The purpose of the first experiment is to compare computational cost of the protocol execution based on the key concept of Meadows' cost-based analysis between all principals with some possible ranges of puzzle difficulty (k) including $k = 0$ (which means that no puzzle is required), easiest value $k = 1$ for contrasting the difference between ad3 and ad4, intermediate values $k = 10$, $k = 20$, and $k = 40$ for a hardest value as instructed in HIP specification. In this simulation, we allow individual initiators to initiate a request token only once, while the responder is able to flexibly adjust the puzzle difficulty within initially defined values. Once the

³More formal descriptions are available on the official website of CPN Tools, <http://wiki.daimi.au.dk/cpntools/cpntools.wiki>

Table 2: Comparison of Computational Cost of HIP with $k=1$ and $k=10$

Authentication Protocol	Initiator			Responder		
		$k=1$	$k=10$	$J, E1, sig_I$ valid	only J valid	Everything invalid
HIP	hc	19973	22017	19591	-	-
	$ad1$	0	0	-	-	2
	$ad2$	14982	17026	19591	-	-
	$ad3$	4	2048	-	4998	-
	$ad4$	0	0	-	-	6

simulation has arrived to the final step, we record the total computational cost of individual user comparing to the responder on specified ranges of k .

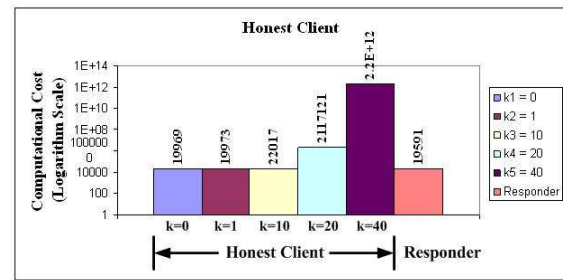
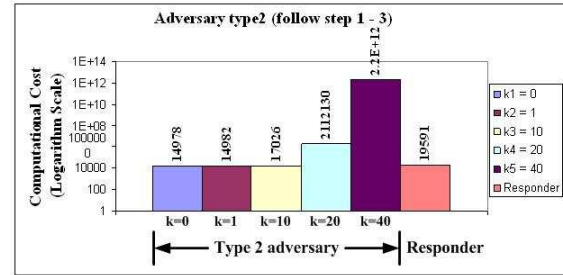
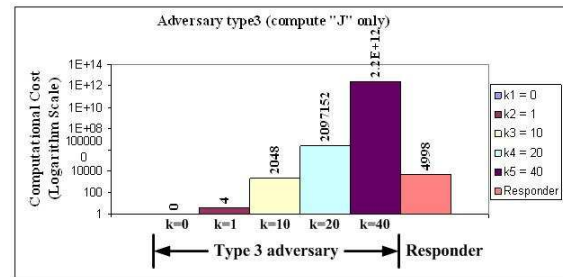
Experimental Result: During the protocol execution, the initiator sends a request message to the responder using the host identity tag (*HIT*) which is a hash of the host identifier (*HI*) used in HIP payload and to index the corresponding state in the end hosts. Therefore, the initiator only employs cheap operations at the beginning step. We assume that the computation at this step can be precomputed, so the cost at the first operation would be negligible. Once the responder receives the requested message, the responder requires a hash operation and some values from the precomputation for returning to the initiator in the second step. This operation is a cheap operation similar to the initiator's.

When the initiator receives the replied message, only honest clients participate in the verification of *HIT* and responder's signature, so the cost is equal to the *HIT* verification plus signature verification. In the case of *ad1*, it does not take any further actions after the first message, therefore the computational cost is zero for the second stage. The operations in message three of the initiator include the brute-force search to find the puzzle solution, and the key generation. The cost of solving a puzzle depends on the value of k in the puzzle message field. However, only an *hc*, *ad2*, and *ad3* is required to solve the puzzle solution. Like *ad1*, the *ad4* does not attempt to solve the puzzle. As a result the puzzle difficulty does not affect computational cost on this type of adversary. Another important thing to note is that, the cost of the adversary to spoof, insert, or interrupt the message has not been defined in this phase. So, we set the cost of generating randomly chosen messages in the case of *ad3* and *ad4* to be zero.

Considering the task on the responder's machine when it receives the third-step message from the initiator, the responder begins to validate the puzzle solution which is defined as a cheap operation because the responder performs only one hash calculation. If it is invalid, the process will stop and the responder will drop the corresponding packet from the connection queue (the system will return a resource to the responder). Otherwise, the responder performs the decryption to obtain an initiator's public key. The responder finally verifies the signature by using the initiator's public key obtained in the previous step. The result would be either valid or invalid. After the authentication has been completed, the responder and the initiator will perform a key confirmation and start to exchange information. Table 2 summarizes the computational cost when the puzzle difficulty is set to $k=1$ or $k=10$ comparing between every principal (honest client and adversaries) and the responder. The experimental result shows that the most effective adversary is *ad3* (the greatest different threshold be-

tween *ad3* and the responder) because *ad3* can force the responder to engage in the expensive tasks, i.e. digital signature verification.

Figure 3 illustrates the computational cost of *hc*, *ad2*, and *ad3*, respectively. In the comparison charts, we measure the cost of those users who engage in solving the puzzle of the difficulty level $k = 0, 1, 10, 20, 40$.

(a) Computational Cost between *hc* and a Responder(b) Computational Cost between *ad2* and a Responder(c) Computational Cost between *ad3* and a ResponderFigure 3: Comparison of Computational Cost on HIP with different ranges of k

Comparison between Figures 3(a) and 3(b) shows that *hc* and *ad2* incur similar computational costs for the same value of k chosen. This illustrates well the effectiveness of HIP in achieving its aims in resisting DoS attacks, at least against this type of adversary. On the other hand, *ad3* and *ad4* spend very small computational resources compared with the responder because both adversaries use some random message elements. This situation would bring the responder to the risk of DoS attacks if the value of k is not chosen properly. Figure 3(c) indicates that a value

of k a little above 10 would be appropriate to ensure that **ad3** uses as much computation as the responder.

Experiment 2: The purpose of the second experiment is to measure a tolerance of the responder and to identify the most effective strategy to deny service. The result can be used to compare with the first experiment for confirming whether Meadows' cost-based framework is effective for evaluating the DoS-resistant protocols or not. In order to examine the protocol, we select two possible values from the experiment 1, i.e. $k = 1$ and $k = 10$ for achieving this simulation. We choose those two values because we want to investigate the different behaviour of **ad3** and **ad4** (if we choose $k = 0$ no client puzzle is required, the task of both adversaries will be identical). Also, both of those values do not put excessive computational effort to the honest client and the total task is still in the acceptable threshold comparing to tasks on the responder (see Figure 3(a)).

In order to allow the responder to flexibly adjust puzzle difficulty between those two values more efficiently, we simply insert a counter into the model for measuring the condition of a responder's workload. Once the workload has reached the maximum tolerance, the responder will increase the puzzle difficulty to the higher level for delaying the incoming rate of requested messages.

Additional to the first experiment, we allow a responder to participate with a pair of initiators (**hc** and a single type of adversary). We assume that a responder has to deal with different strategies of adversary and different amount of packets which consist of both legitimate and bogus messages. With regard to the number of packets, we allow three different scenarios for both honest client and adversary;

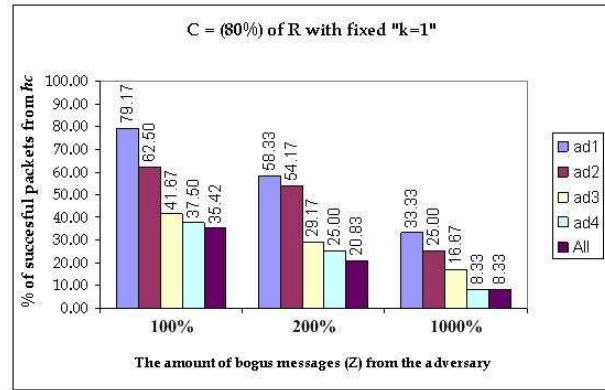
- *Honest Client*: initiates requests, C , at 80%, 100%, and 150% of the responder's capacity (R).
- *Adversary*: floods bogus requests, Z , at 100%, 200%, and 1000% of the responder's capacity (R).

Considering the initiator's packet, there are different actions from initiators and the responder during the protocol run. Honest clients initiate a request only once and keep waiting to process next steps. This delay is described by means of Timed CPNs, i.e. every transition which relates to cryptographic operations is defined as a timed process. This amount of time is calculated and assigned by using data from cryptographic benchmarks shown in Table 1. During the simulation, if honest client's requests have been rejected under DoS circumstances, **hc** gives up to open another session. In term of adversaries, there are two different situations when their packets are rejected by the responder; 1) the responder detects the bogus messages during the verification steps, and 2) the responder does not have enough resources for serving any requests. Once the responder detects the attack and rejects those packets, adversaries will lose those packets from the system.

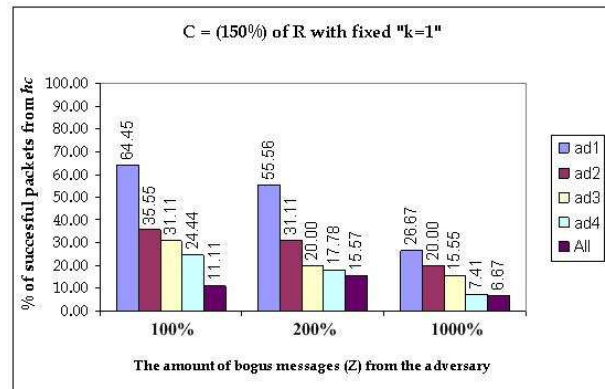
Finally, in order to examine the tolerance of HIP protocol under different attack strategies, each individual adversary has been made a pair with an honest client during the protocol execution. To make an evaluation, the number of successful legitimate requests that the responder can serve under different adversary's abilities has been measured as the percentage of successful packets. We achieve this task by inserting places for displaying the number of completed and rejected messages at the responder's network.

Experimental Result: In the second experiment when we prohibit the responder's ability to adjust k ,

we have seen from Figure 4 that when adversaries increase the number of bogus messages in the system, the percentage of successful messages from honest clients to obtain a service will drop drastically. Comparing different types of adversary, the most effective is **ad4** who sends bogus messages to the responder by crafting messages randomly. This is because **ad4** can flood a large number of messages to overwhelm the responder's resource quicker than the others, which causes the responder to reject the next incoming messages from honest clients. Although adversaries' packets will be detected and discarded by the responder, all adversaries are able to flood new bogus messages as soon as they receive returned packets from the responder at phase two.



(a) hc's load = 80% of R



(b) hc's load = 150% of R

Figure 4: Percentage of throughput from **hc** with $k=1$

Comparing **ad1** and **ad4**, even though both of them craft random messages, **ad4** achieves its goal at a higher rate than **ad1**. That is because the responder can process the incoming request at step 1 and clear a connection queue faster than step 3. (In step 1, the responder only participates in the protocol by choosing the puzzle difficulty (k) and pre-computed information, and then returns it to **ad1**, which involves an expensive verification in step 3.) Although **ad1** can re-send bogus messages after receiving replied messages, this does not cause the responder to reject a large number of messages because HIP mitigates the problem of flooding messages to overwhelm a resource at step 1 by adopting a stateless-connection. On the other hand, the task of **ad4** to fill up the responder's queue at step 3 by flooding random messages can be achieved more easily than **ad1**. Even though HIP integrates a gradual authentication in this step, the process of checking a puzzle solution and a digital signature is longer than the whole process at step 1, therefore, the responder's queue would be more easily overwhelmed in step 3 than in step 1.

Considering **ad2** and **ad3** who attempt to deny services at phase 3 by computing the puzzle solution, the result shows that **ad3** succeeds at a higher proportion than **ad2**. The reason is that **ad3** floods attack messages at higher speed than **ad2** who uses more effort in the generation of message two than **ad3**. Nonetheless, both of them can force the responder to engage in signature verification. (Although **ad4** floods a large number of messages at step 3 as well as **ad2** and **ad3**, **ad4** cannot force the responder to engage in expensive operations because the responder is able to detect the forgery at the cheap phase; the puzzle verification.)

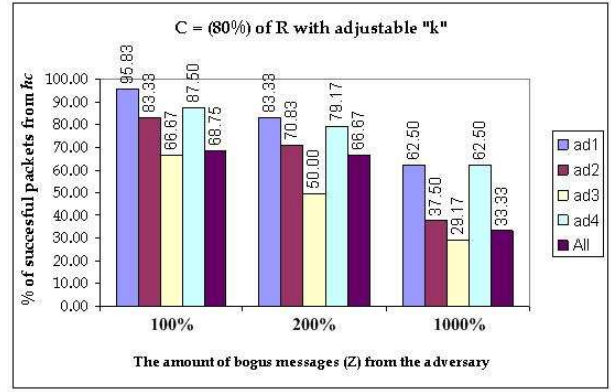
Without the ability to adjust puzzle difficulty, the percentage of successful messages in the system with **hc** and **ad4** is lower than the others because **ad4** floods message three at a higher rate than the other types. As a result, the most effective adversary to deny service to the responder would be **ad4** that attacks the verification phase. Particularly, most key agreement protocols incorporate verification tasks that would be susceptible to resource exhaustion attacks.

Finally, the result of the combination attack, illustrated as **A11** in the graph of figure 5, shows that when the responder has to deal with all types of adversary, the percentage of legitimate users served by the responder will fall significantly with increment of bogus messages.

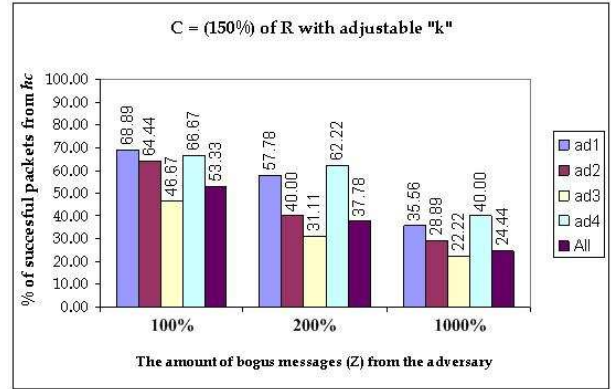
Having analysed the system with non-adjustable client puzzle, we would like to compare such results to the system integrating with adjustable client puzzle for investigating the usefulness of puzzle difficulty under equivalent magnitude of resource exhaustion attacks. In order to adjust the puzzle difficulty, we allocate two possible values for the responder to determine. Under normal circumstances, the responder selects $k=1$, which means an easiest puzzle solution is required from the initiator. Once the responder receives more requested packets than its maximum capacity to handle, the responder raises the puzzle difficulty to a next level that we set to $k=10$. This parameter would help the responder to slow down the incoming rate by requiring work of the initiator to solve puzzles at the factor of 2^{10} because this puzzle technique is Aura's hash-based puzzle (Aura, Nikander & Leiwo 2000).

From the experimental result in figure 5, the number of attacking machines that the responder can tolerate is increased, i.e. honest clients have succeeded to establish a connection at higher proportion compared to the result of an experiment which includes attack techniques from four types of adversary.

Another interesting result is that the successful rate of message from **hc** in the case of **ad4**'s strategy is higher than the non-adjustable technique. The reason is that **ad4** does not compute the puzzle solution, so, no matter what the puzzle difficulty is, **ad4** can flood bogus messages at the equivalent speed as the simulation of a non-adjustable puzzle. However, at that amount of bogus messages, there are only messages from **ad4** (no legitimate traffic because honest clients have to spend some amount of time to solve the puzzle solution), or only a few messages from **hc** that arrive to the connection queue before the responder increases puzzle difficulty. As a result, the responder can validate the puzzle solution before the next group of messages have arrived. Undoubtedly, these bogus messages from **ad4** will be rejected at the first step of verification which requires only a short period and removes such attack from the connection queue. However, this situation does not occur in the case of **ad3** because **ad3** has to spend time to solve the puzzle as well as **hc**.



(a) hc's load = 80% of R



(b) hc's load = 150% of R

Figure 5: Percentage of throughput from **hc** with k is chosen between 1 and 10

5 Discussion

Comparing the system when the responder is unable to adjust puzzle difficulty, the experimental result from Meadows' cost-based framework identifies that **ad3** is the most effective strategy (Table 2 in the column $k=1$) because **ad3** spends tiny computational cost to force the responder to compute up to 4998 computational units. (Although **ad4** does not spend any computation, the responder requires only a cheap computation for detecting and discarding those bogus messages.) In contrast, the Timed CPNs simulation indicates that **ad4** is the most effective strategy shown in Figure 4.

When we allow adjustable puzzle difficulty (Figure 5), **ad3** becomes the most effective technique in the Timed CPNs simulation-based analysis, while **ad4** is identified as the most effective technique in Meadows' cost-based framework. (Once the responder slightly increases puzzle difficulty, the different gap of computational cost between **ad3** and the responder becomes narrow, but remains the same between **ad4** and the responder. The cause of this result from **ad4** is already explained in Section 4.3.)

The reason why both simulations provide us a different result is because we allow the adversaries' capability to constantly flood new bogus messages when they receive return messages at the second phase in the time-based analysis. As a result, **ad4** should be the most effective adversary who depletes the responder's connection queue and blocks legitimate users from acquiring services faster than others. Comparing to **ad3**, even though those bogus messages from **ad4** will be removed from the connection queue quicker, a new stream of **ad4**'s bogus messages still keeps the responder busy to verify them.

Contrasting the experimental result of Meadows'

cost-based framework and the Timed CPNs simulation, the cost-based analysis considers only the cost to an individual adversary to perform a set of identified actions to a certain point of the protocol execution compared to the cost of protocol engagement by the responder's machine. The simulation-based analysis is able to define more sophisticated techniques of the adversary as well as to involve a large amount of packets launched by multiple participants. Moreover, some hidden parameters, such as *time* as examined by Chan et al. (2006), can be included and evaluated in the analysis. As a result, lack of ability to model realistic events related with time factors and to handle large amount of messages is a limitation for analysing DoS-resistant protocols in Meadows' cost-based framework.

Finally, the most obvious benefit from the simulation-based analysis is that we can observe not only the behaviour of adversaries, but also the consequence of attacks to the system during the protocol execution. In addition, the analyst is able to more easily understand and evaluate the final outcome contrasting with cost-based analysis. In the cost-based evaluation, the analyst has to take extra care in order to consider and identify adversary capability as well as the effect of such attacks because sometimes the cost-based framework might generate ambiguous output; for example, considering the result of **ad3** comparing with **ad4** under cost-based analysis from Table 2, we conclude that the most effective strategy is performed by **ad3** because the total cost difference between the responder and such adversaries is the most. However, some may argue that **ad4** should be the most destructive scenario because the ratio of computational effort between the responder and such adversaries is infinity no matter what the chosen value of puzzle difficulty is.

6 Conclusion

This work has achieved the aims of extending the Meadows' cost-based framework to provide more accurate representation of computational cost and shown the potential of automated analysis. Moreover, we have explored unbalanced computational vulnerabilities of HIP by using a simulation approach provided in CPN Tools. By comparing experimental results from Meadows' cost-based and simulation-based analysis, we have found a limitation of Meadows' framework to define the ability of advanced adversaries and address DoS threats.

Because the nature of DoS attacks is quite subtle, the protocol developer should take extra care in the design and evaluation phase. Simulation is one promising technique that provides state exploration for searching all possible vulnerabilities in DoS-resistant protocols. In future work, we plan to extend this research by using the model checking capabilities of CPN tools to automatically verify the system by traversing the model and checking whether the cost tolerance between initiator and responder exceeds some reasonable threshold. Moreover, the power of adversaries can be extended in different ways in order to model more powerful attacks. For example, the advanced adversary, who attempts to attack the protocol at the third message, can be extended to flood reused packets from previous connections, eavesdrop messages from a valid communication, or craft bogus messages using existing messages including valid or invalid puzzle solutions as well as digital signatures. When inserting such advanced abilities into the model, we also require a technique to measure and identify cost of those operations in order to achieve a formal analysis.

References

- Aiello, W., Bellovin, S. M., Blaze, M., Ioannidis, J., Reingold, O., Canetti, R. & Keromytis, A. D. (2002), Efficient, DoS-resistant, secure key exchange for internet protocols, *in* 'The 9th ACM Conference on Computer and Communications Security', ACM Press, Washington, DC, USA, pp. 48–58.
- Al-azzoni, I. (2004), The Verification of Cryptographic Protocols using Coloured Petri Nets, Master of Applied Sciences Thesis, Department of Software Engineering, McMaster University, Ontario, Canada.
- Aura, T., Nagarajan, A. & Gurtov, A. (2005), Analysis of the HIP Base Exchange Protocol, *in* 'the 10th Australasian Conference on Information Security and Privacy (ACISP 2005)', Vol. 3574 of *Lecture Notes in Computer Science*, Springer-Verlag, Brisbane, Australia, pp. 481 – 493.
- Aura, T. & Nikander, P. (1997), Stateless Connections, *in* 'International Conference on Information and Communications Security', Springer-Verlag, Beijing, China, pp. 87–97.
- Aura, T., Nikander, P. & Leiwo, J. (2000), DoS-resistant authentication with client puzzles, *in* 'Security Protocols Workshop 2000', Cambridge, pp. 170–181.
- CERT (1996a), Denial-of-Service Attack via ping. [Online]. Available: <http://www.cert.org/advisories/CA-1996-26.html>.
- CERT (1996b), TCP SYN Flooding and IP Spoofing Attacks. [Online]. Available: <http://www.cert.org/advisories/CA-1996-21.html>.
- Chan, M. C., Chang, E., Lu, L. & Ngiam, P. S. (2006), Effect of Malicious Synchronization, *in* '4th International Conference on Applied Cryptography and Network Security (ACNS'06)', Singapore, pp. 114–129.
- Dai, W. (2004), Crypto++ 5.2.1 Benchmarks. [Online]. Available: <http://www.eskimo.com/~weidai/benchmarks.html>.
- Doyle, E. M. (1996), Automated Security Analysis of Cryptographic Protocols using Coloured Petri Net Specification, Master of Science Thesis, Department of Electrical and Computer Engineering, Queen's University, Ontario, Canada.
- Gupta, V., Gupta, S., Chang, S. & Stebila, D. (2002), Performance Analysis of Elliptic Curve Cryptography for SSL, *in* 'WISE'02: Proceedings of the 3rd ACM Workshop on Wireless Security', ACM Press, Atlanta, GA, USA, pp. 87–94.
- Han, Y. (1996), Automated Security Analysis of Internet Protocols using Coloured Petri Net Specification, Master of Science Thesis, Department of Electrical and Computer Engineering, Queen's University, Ontario, Canada.
- Jakobsson, M. & Juels, A. (1999), Proofs of work and bread pudding protocols, *in* 'the IFIP TC6 and TC11 Joint Working Conference on Communications and Multimedia Security (CMS 99)'.
- Jensen, K. (1997), *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, Vol. 2nd edition, Vol. 1-3, Springer-Verlag.

- Jensen, K. (1998a), A brief introduction to Colored Petri Nets, in 'Workshop on the Applicability of Formal Models', Aarhus, Denmark, pp. 55–58.
- Jensen, K. (1998b), An introduction to the Theoretical Aspects of Colored Petri Nets, in 'Workshop on the Applicability of Formal Models', Aarhus, Denmark.
- Juels, A. & Brainard, J. (1999), Client Puzzles: A Cryptographic Defense Against Connection Depletion Attacks, in 'the 1999 Network and Distributed System Security Symposium (NDSS '99)', Internet Society Press, Reston, San Diego, California, USA, pp. 151–165.
- Meadows, C. (2001), 'A Cost-Based Framework for Analysis of DoS in Networks', *Journal of Computer Security* **9**(1/2), 143–164.
- Meadows, C. (2003), 'Formal methods for cryptographic protocol analysis: emerging issues and trends', *IEEE Journal on Selected Areas in Communications* **21**(1), 44–54.
- Moskowitz, R. (2006), 'The Host Identity Protocol (HIP)', Internet Draft, Internet Engineering Task Force. <http://www.ietf.org/internet-drafts/draft-ietf-hip-base-06.txt>.
- Neih, B. B. & Tavares, S. E. (1993), Modelling and Analysis of Cryptographic Protocols using Petri Nets, in 'Advances in Cryptology', Berlin, Germany, pp. 275–295.
- Petri, C. A. (1962), Kommunikation mit Automaten, PhD Thesis, Institut für Instrumentelle Mathematik, Schriften des IIM.
- Ramachandran, V. (2002), Analyzing DoS-Resistance of Protocols Using a Cost-Based Framework, Technical Report YALEU/DCS/TR-1239, Department of Computer Science, Yale University.
- Smith, J., Nieto, J. M. G. & Boyd, C. (2006), Modelling Denial of Service Attacks on JFK with Meadows's Cost-Based Framework, in 'Fourth Australasian Information Security Workshop (AISW-NetSec 2006)', Vol. 54, CRPIT series, pp. 125–134.
- Tan, Z., Lin, C., Lin, H. & Li, B. (2004), 'Optimization and Benchmark of Cryptographic Algorithms on Network Processors', *IEEE Micro* **24**(5), 55–69.
- The Department of Computer Science, University of Aarhus, Denmark (2004), CPN Tools: Computer Tool for Coloured Petri Nets. [Online]. Available: <http://wiki.daimi.au.dk/cpn-tools/cpn-tools.wiki>.

Appendix

A Cost-Based Definition

To analyse the protocol specification, we begin with the notation introduced by Meadows (2001).

Definition 1: The sequence of messages sent from the principal A to the principal B in the protocol is written in the form

$$A \rightarrow B : T_1, \dots, T_k \parallel M \parallel O_1, \dots, O_n$$

- T_i are the operations performed by the principal A for preparing the message M .

- M is the sent message from the principal A and subsequently received by the intended recipient B .
- O_j are the operations performed by the principal B for processing the message M .

Definition 2: An event in the operation $A \rightarrow B : T_1, \dots, T_k \parallel M \parallel O_1, \dots, O_n$ is one of:

- an operation in T_i or an operation O_j .
- A sends M to B , or B receives M from A .

There are three types of events:

1. **Normal events** appear at both principals and always succeed and are followed by the next events.
2. **Verification events** appear only at the recipient side and can evaluate to either success or failure.
3. **Accept events** appear at the end of the operations O_j indicating the completion of the process.

Definition 3: A cost set C is a set with operation $+$ with partial order \geq such that

$$x + y \geq x \text{ and } x + y \geq y \quad \text{for all } x, y \in C.$$

Definition 4: Event cost function (δ) and protocol engagement cost function (Δ)

- δ is an event cost function iff it transfers sets of operations in the protocol into the cost set C which consists of four values: expensive > medium > cheap > 0.
- δ' is a message processing cost function related to δ defined on verification events $\{V_i\} \subset \{O_j\}$ such that

$$\begin{aligned} \text{for } A \rightarrow B : T_1, \dots, T_k \parallel M \parallel O_1, \dots, O_n \\ \text{if } \{V_i\} = \{O_j\}, \text{ then} \\ \delta'(V_i) = \delta(O_1) + \dots + \delta(O_j) \end{aligned}$$

- $\Delta(O_n)$ is the sum of all operational costs appearing at the responder B up to the accept event O_n , plus operational costs of processing the response message related to such accept event O_n .

Definition 5: Attacker event cost function ϕ , attacker cost function Φ , and attack cost function Θ are defined as follows.

- $\Phi(\{x_1, \dots, x_n\}) = \phi(x_1) + \dots + \phi(x_n)$ where x_i are the set of available attacker's capabilities and ϕ is a function for interpreting attacker's capabilities to cost set.
- Θ is a function from the set of events defined by the protocol to a cost set.

Definition 6: Defender cost set C , attacker cost sets G , tolerance relation τ are defined as follows.

- $\tau \subset C \times G$; consists of all pairs (c, g) such that the protocol designer is willing to tolerate the responder expending resources up to cost c , as long as the attacker has to extend resources of cost g or greater. A tuple (c', g') is said to be within the tolerance relation if there exists $(c, g) \in \tau$, such that $c' \leq c$ and $g' \geq g$.

Once the actions of each protocol principal are classified into the computational costs cheap, medium, or expensive, all actions of the protocol run can be compared between the initiator and the responder. The protocol is secure against DoS attacks, if the final cost is great enough from the point of view of the attacker in comparison with the cost of engaging in the events up to an accepted action from the point of view of the defender. Otherwise, we conclude that the protocol is insecure against DoS attacks.

The CRSS Metric for Package Design Quality

Hayden Melton, Ewan Tempero
 Department of Computer Science
 University of Auckland
 Auckland, New Zealand
 {hayden|ewan}@cs.auckland.ac.nz

Abstract

Package design is concerned with the determining the best way to partition the classes in a system into subsystems. A poor package design can adversely affect the quality of a software system. In this paper we present a new metric, *Class Reachability Set Size (CRSS)*, the distribution of which can be used to determine if the relationships between the classes in a system preclude it from a good package design. We compute CRSS distributions for programs in a software corpus in order to show that some real programs are precluded in this way. Also we show how the CRSS metric can be used to identify candidates for refactoring so that the potential package structure of a system can be improved.

1 Introduction

The classes in an object-oriented software system can be partitioned into groups, or *subsystems* (Wirfs-Brock, Wilkerson & Wiener 1990, p.135). These subsystems serve to provide a higher-level view of the key abstractions in the system than that which is represented by individual classes (Booch 1991). In large-scale software systems, comprising thousands of classes, subsystems are absolutely essential (Lakos 1996, Booch 1991, Coad & Yourdon 1991, Martin 1996b). They help us to avoid information overload — a result of the limits on the human mind's information-processing capacity (Coad & Yourdon 1991). They facilitate a vocabulary that developers of a system can use in communication (Coad & Yourdon 1991, Booch 1991). They allow managers to determine a *partial ordering* of development activities with respect to time, that allows for parallelism in development effort (Meyer 1995, Lakos 1996, Martin 1996b). Finally, they have a significant impact on the system's quality particularly with respect to reusability and testability (Lakos 1996, Martin 1996b, Szyperski 1998).

One of the challenges in partitioning classes into subsystems is that for any given set of classes there are many possible ways to partition them (Martin 1996b). The choice of partitionings is strongly influenced by the classes that make up the system because it is the relationships between these classes that cause relationships between the partitions in a given partitioning. It follows that relationships between partitions can be altered by moving classes between partitions (repartitioning) or by altering the source code of these classes to break their relationships with other classes.

Package design is the area concerned with determining the 'best' way to partition classes into subsystems (Martin 1996b). The question addressed in this paper is "do the relationships between the classes in a system preclude them from a 'good' partitioning?". In order to answer this question we must determine what constitutes

a 'good' partitioning. One contribution of this paper is a careful discussion of how partitioning (or package design) purportedly affects the external quality attributes of *reusability* and *testability*. In the case where the relationships between classes in a system do preclude them from a good partitioning we also provide advice on how to improve this situation through a refactoring strategy.

We define a metric, *Class Reachability Set Size (CRSS)*, which we use in order to determine if the relationships between the classes in a system preclude them from a good package design. This metric counts, for a given class, all the other classes in the system's source code that it transitively depends-on for its compilation. In this way the metric takes into account the *whole system*, not just individual classes from selected subsystems. We show how the *distribution* of the CRSS values for all of the classes defined in system is useful for answering our question. We present empirical evidence to support this claim.

Our use of the CRSS metric is to determine whether design principles proposed by others can be met by an existing class structure, that is, we provide an *operational* means to check conformance to these design principles. However, the CRSS metric says nothing about whether the design principles themselves are correct, that is, following the principles lead to a higher quality design than not following them. In fact, we have found very little empirical evidence to support *any* design principles. We consider this a serious lack in software engineering research. We believe this is due to the difficulty in operationally checking conformance, and so believe that developing metrics such as CRSS to be a promising approach to understanding the structure of software.

The remainder of this paper is organised as follows. In Section 2, we survey the package design principles proposed in the literature. Section 3 discusses in detail how these package design principles impact testability and reusability. We then present the CRSS metric in Section 4. In Section 5 we present an empirical study on the use of CRSS on a corpus of Java software systems. In Section 6, we demonstrate a new refactoring strategy that uses CRSS and one other metric in order to identify classes for refactoring so the package design quality can be improved. We discuss related work in Section 7 and conclude with Section 8.

2 Background

2.1 Package Design

Programming languages such as Java, C++ and Ada support a higher-level of organisation through the package construct. Packages allow classes to be organised into named abstractions more generally referred to as *subsystems*. Within a system there may be subsystems at several levels of abstraction (Lakos 1996, Wirfs-Brock et al. 1990, Coad & Yourdon 1991, Booch 1991). In this respect the subsystems at a given level of abstraction can

themselves be partitioned into new subsystems representing a higher-level abstraction.

Package design is ultimately about organising classes into subsystems. In this respect programming languages without the package construct can still allow the level of abstraction provided by subsystems. Subsystems can be realised without the use of packages by arranging source files into separate (file system) directories, the names of which identify these subsystems. Alternatively, or additionally, subsystems can be realised through multiple class declarations in a single source file (Lakos 1996).

Many authors have identified principles for package design but have referred to it using different terms. Lakos, for instance, uses the term *physical design* to collectively refer to his principles for package design (Lakos 1996, p.97). Martin uses the term *package design* (Martin 1996b). Earlier work in package design often does not give it an explicit name but uses terms such as *class category* (Booch 1991), *clusters* (Meyer 1995), *subject areas* (Coad & Yourdon 1991), *domains* (Shlaer & Mellor 1992) and *subsystems* (Booch 1987) to refer to its fundamental units. Our review of the package design literature has identified two flavours of design principles. First are those that relate to the formation of classes into individual packages. Second are those that relate to properties of the directed graph formed by the dependencies between the packages at a given level of abstraction.

2.1.1 Package Formation

The package construct, at least in Java, is a recursive structure in that it contains classes and/or other packages. It is the recursive nature of a package that allows it to represent subsystems at different levels of abstraction. A given package can represent a subsystem at a higher level of abstraction than the subsystems represented by the packages it contains. The principles of **manageable size**, **stand-alone**, **cohesive**, and **encapsulation** have been proposed to guide package design. We address the first two in this paper.

Manageable Size. The number of items (packages or classes) contained by a package should not exceed a given limit. Coad et al. identify Miller's paper on 'the magic number seven, plus or minus two' (Miller 1956) as the basis for this principle (Coad & Yourdon 1991, p.107). Essentially Miller's paper states that the short-term memory of a human can hold 5-9 things at a time. Based on Miller's work it could be argued that for package to be quickly understood (using our short-term memory) it should contain 5-9 other packages or classes. Other authors differ on this limit, but nevertheless identify the need for a limit to a package's size. Lakos identifies 500 to 1000 lines of code (LOC) for a *component* (low-level subsystem), and 5000-50000 LOC or a few dozen components per *package* (higher-level subsystem) (Lakos 1996, p.481). Meyer states a *cluster* should contain 5-40 classes and be able to be developed by 1-4 people and entirely understood by a single person (Meyer 1995, p.51).

For the purposes of this paper we will not specify a particular limit for package size other than to say that such a limit should exist, and that the limit can be stated in terms of number of classes directly or indirectly contained by a package. The limit may be dictated by company policy, personal preference, or some other mechanism. All of our arguments apply to any limit on package size, so long as the limit exists.

Stand-alone. A package should be stand-alone in that it should have minimal dependency on other packages (Lakos 1996, p.147). A given package depends on another if its classes cannot be compiled without some of the latter's classes. The notion of compilation dependency among packages is important because we want to be able to lift packages from one program for deployment in another. In this way we can reuse code without having to

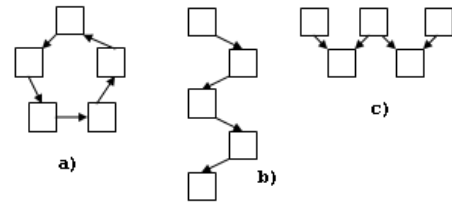


Figure 1: Cyclic, tall and flat PDGs

modify its textual content to remove dependencies. The stand-alone property of a package is also important for understandability, testability, and the extent to which parallel development effort can occur across packages in a system (Lakos 1996, p.149-202).

2.1.2 Graph Properties

The principle that a package should be stand-alone leads to more package design principles when it is applied to *all* the packages in a system. These principles are auxiliary in that certain characteristics of what we refer to as *Package Dependency Graphs (PDGs)* imply that a system's packages are not stand-alone. If a system's PDG contains cycles, or is 'taller' rather than 'flatter', then the packages that comprise the system cannot be as stand-alone as their flat, acyclic analogs.

A PDG is a directed graph representing all the packages in a system's source code at a given level of abstraction as nodes, and compilation dependencies between these packages as directed edges. Packages have compilation dependencies on each other due to the underlying dependencies between the classes that they contain (both directly and indirectly through their subpackages). We say package A *depends on* package B if any class directly or indirectly contained by A depends on any class directly or indirectly contained by B. We will present a formal definition of what it means for a class to depend on another in Section 4. Also, since packages may exist at different levels of abstraction in a system, a system may have several PDGs.

The graphs in Figure 1 are PDGs. We can reasonably compare them to one another because they comprise the same number of subsystems (vertices) and the same number of dependencies (edges) (except (a), which has an extra edge). The purpose of these PDGs is to illustrate that tall and cyclic graphs cannot comprise packages that are stand-alone, so PDGs should be flat and acyclic. Consider firstly any package from the cyclic PDG 1(a). In order to deploy this package in another program we also have to copy with it all the other packages in the graph. Even though the package itself directly depends on only one other package, this other package also depends on another to compile. The process goes on for the transitive closure of the dependency so at least in terms of deployment the stand-alone property of a package can be considered on the basis of *transitive* dependencies.

The argument is similar for the tall graph of Figure 1(b) the top-most package requires all other packages in order for it to be deployed in another system. The tall graph of (b) is better than in the cyclic graph of (a) because packages towards the bottom of the graph transitively depend on fewer and fewer other packages. The flat graph of Figure 1(c) is better than the tall and cyclic graphs because it has the most packages that can be deployed with the minimal number of other packages, so each package is more 'stand-alone'.

One problem with the PDGs of Figure 1 is that they are not indicative of real designs because real designs tend to have more direct dependencies between packages and tend to have more 'layers'. Lakos claims that a PDG that forms a balanced binary tree (see Figure 2) is a good reference point with which compare real designs (Lakos 1996,

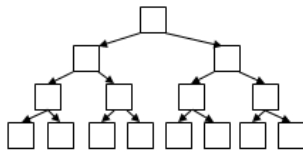


Figure 2: Balanced Binary Tree PDG

p.187), although he notes that real designs are not nearly so regular. In terms of deployment, leaves of the tree are the most stand-alone since they depend on no other packages. More than half of the packages in a balanced binary tree depend on no other packages. One quarter of the packages in such a tree can be deployed with just two other packages, and so on.

This discussion of design principles for PDGs has justified these principles in terms of the packages in a graph being stand-alone so that they can be deployed in other systems. There are more reasons other than deployment for ensuring that PDGs are flat, acyclic graphs. These are more complicated and are discussed in the following section.

3 Effects on Quality

The motivation for any design principle, whether it relates to classes (e.g. group related logic and data together), object interactions (e.g. design patterns) or packages is that the application of the principle will improve the quality of software system in some way. With regard to package design the claim is that allocating classes to packages according to the package design principles described above will result in a system of higher quality than if classes were allocated to packages in a more ad-hoc fashion. In this section we present a discussion of how the manageable size and stand-alone package design principles clearly relate to reusability and testability.

3.1 Reusability

Reusability is defined as “the degree to which a software module or other work product can be used in more than one computer program or software system” (IEE 1990). One can reuse things of a conceptual nature such as software architecture descriptions and design patterns, or things of a more ‘binary’ nature such as procedures, classes and modules (Szyperski 1998, p.3-4). The literature on package design claims improved reusability on the basis of reusing the functionality implemented in the source code of one program in another. This relates to quality because reusing code from one program in another can lead to reduced development effort and fewer defects since the reused code has been ‘proven’ in the context of its original program (Gaffney Jr & Cruickshank 1992).

Code reuse involves copying source files (and the libraries that they depend on) from one program to another, without having to modify the textual content of these source files. Compare this to *code copying* where text is copied from one program to another, usually meaning the copier has to modify the text to make it work in his environment and the copied code eventually diverges so much from the original that it becomes unrecognisable. This is a problem because the copier of the code becomes responsible for its implementation and it is no longer possible to easily integrate new versions of the code from its origin system following bug fixes and enhancements made by the owner of the code (Martin 1996b).

Packages are inherently related to code reuse because a class is not the fundamental unit of deployment (Martin 1996b, Lakos 1996) (Szyperski 1998, p.10). It would be unusual for a single class copied from one program to be able to be compiled in the context of another program.

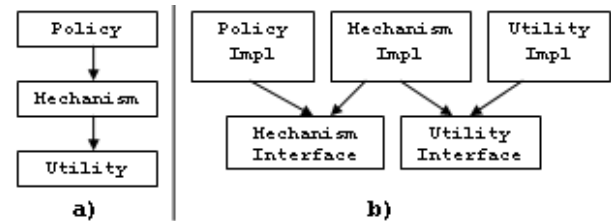


Figure 3: Flattening a tall PDG with the DIP

Chances are that the class would depend on other classes appearing in its methods return types and parameters, as well as in the bodies of its methods’ implementations. If the class performed some domain-specific function then it is likely that at least some of the classes it depends on would also be defined in other source files of the original program (as opposed to classes defined in the programming language’s API). In this way whole packages should be copied (Martin 1996b), not individual source files each containing a single class.

In terms of package design, packages that are stand-alone (and of a manageable size) lend themselves to reuse because we have to copy fewer packages (and classes) from one system to the other. While the sheer number of the packages copied is of concern because it increases the amount of code in the system that needs to be understood and can contain bugs, it is not the main problem. Rather the problem is that many of the packages are not likely to be strictly necessary for the given package to provide its functionality, or as Lakos (Lakos 1996, p.14) states “in order for a ... subsystem to be reused successfully, it must *not* be tied to a large block of unnecessary code”. This is where flatter, acyclic graphs come in.

Tall or cyclic PDGs are not well suited towards reuse and many such graphs can be transformed to become flatter and acyclic through class refactoring and splitting packages into an ‘implementation’ and an ‘interface’. Martin’s *Dependency Inversion Principle (DIP)* shows how a tall graph can be transformed into a flat graph (Martin 1996a). Figure 3 illustrates the transformation proposed by Martin. The individual packages in Figure 3 have an improved potential for reuse because the ‘implementation’ packages are more *flexible*. They are more flexible because they can be used with different implementations of the other packages. For instance the Mechanism Implementation package in 3(b) can now be used with different implementations of the Utility interface. This also means that the reliability of the packages in 3(b) is improved because we do not have to deploy the implementation packages in another system if we do not want them. For instance we can deploy Mechanism’s implementation in a new system without having to copy Utility’s implementation into the system. Not having to copy the Utility implementation can improve the reliability of the new system because Utility implementation cannot be a further source of bugs if it does not exist in the system.

3.2 Testability

Testability is often defined as the ease at which software can be made to demonstrate faults through testing (Bass, Clements & Kazman 1998, p.88). Package design purports improved testability by demonstrating faults through execution driven by automated unit tests (as opposed to execution driven by a user) (Lakos 1996).

We define an automated unit test as a piece of code that exercises another piece of code, and automatically compares the expected effect of that execution to the actual effect in order to report success or failure of that test. This type of testing is particularly useful for regression testing i.e. identifying faults that have been caused by unintended

effects of a modification to a system outside its apparent scope.

Flat, acyclic PDGs have subsystems that lend themselves well to automated unit testing for reasons similar to why they lend themselves to reuse. If a subsystem in a PDG depends on an interface rather than an implementation we can more easily test it using stubs (or *mock objects*, as they are more popularly referred to nowadays). Stubs increase *controllability* and *observability* during testing (Binder 1999, p.980). In terms of controllability we can implement a stub to exercise the boundary values and special cases for the given subsystem's interactions with the package it depends upon. This is essential when these special cases occur as a result of nondeterministic behaviour, or are difficult to set up, or are difficult to trigger (e.g. an out-of-disk-space error) or have callback functions (Thomas & Hunt 2002) in the dependee package's actual implementation.

Flat, acyclic PDGs with stand-alone components of a manageable size are also more cost effective to unit test because they can be tested in 'isolation' (Lakos 1996). This relates back to testing a subsystem using stubs, rather than the actual implementations it depends on at runtime. The rationale for this claim is that testing in isolation means that stubs and test cases are created just to test the functionality provided by the component itself. This means that the complexity of the test reflects the complexity of the component. Reducing the complexity of the test is important because units tests are also code which costs money to produce. A further advantage of testing in isolation is that the tests provides a small but comprehensive example illustrating the use of that subsystem, helpful to someone wanting to reuse it (Lakos 1996).

3.3 Other Quality Attributes

Other claims have been made regarding the effect of package design principles on quality. Coad and Booch imply package design improves *buildability* by facilitating a vocabulary that developers of a system can use in communication (Coad & Yourdon 1991, Booch 1991). Another claim is that package design allows managers to determine a *partial ordering* of development activities with respect to time, that allows for parallelism in development effort (Meyer 1995, Lakos 1996, Martin 1996b). Probably the most contentious claim is that package design principles lead to a package structure that makes the software system more understandable. While it is clear that packages provide a higher level of abstraction than classes which helps us to avoid information overload (Coad & Yourdon 1991), it is less clear whether the 'interface' and 'implementation'-style of packages particular to flat, acyclic PDG improve understandability because they seem to increase the number of packages in the system (compare Figure 3(a) to (b)).

4 Class Reachability Set Size

The *Class Reachability Set Size (CRSS)* metric is computed from the *Class Dependency Graph (CDG)*. A CDG is a directed graph where the vertices are the top-level classes defined in the source files of the software system and the edges represent compilation dependencies. The CRSS for a class is then the number of vertices reachable from the vertex representing that class.

More formally, for a class C , the relation $\text{DEPENDS-ON}(C)$ is the set of classes that must be available in order to compile C (ignoring those classes referred to by redundant `import` statements). In practical terms, in Java, it is the set of `.class` files that must be on the classpath in order to compile $C.java$. Another way to think about it is that is the number of distinct types that are referred to by names that appear in $C.java$. $\text{CRSS}(C)$ is then the

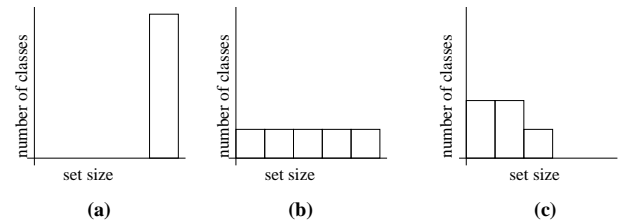


Figure 4: Relationship between PDG structure and CRSS distributions

size of the set representing the *transitive closure* of the DEPENDS-ON relation as applied to C .

Our interest is in the best possible package structure allowable with respect to packages being stand-alone and of manageable size given the relationships between the classes in the system. Just considering the measurements given by CRSS for a single class is not going to do this. What we need is something that is representative of the whole system, not just an individual element of it. Rather than consider something like the mean or standard deviation of CRSS, we use the *distribution* of the CRSS values. As we will argue below, it is the *shape* of this distribution that is important in understanding the best potential quality of a package design for the system.

Since we are computing CRSS for every class in the system, we need to be clear as to which classes' CRSS values are used in the distribution. We only consider 'top level' classes, that is, those that are not nested. Nested classes are not directly represented in the distribution, although their DEPENDS-ON set is computed and contributes to the CRSS value of their lexically enclosing class.

Nested classes are not represented in the CDG because their use seldom constitutes a major design decision (Booch 1991, p.161). It is often the case that these classes are not visible outside their lexically enclosing top-level class, which means other classes cannot depend on them. The classes on which a nested class depends are merged with the dependencies of its top-level class in order not to perturb the actual dependencies between packages. This is assuming that a nested class belongs to the same package as its top-level counterpart, which is certainly true for Java.

We also consider only classes defined in the source files of a system because these are the only classes within the system whose package membership can be altered. Classes defined in external libraries are often in binary (vs. source) form, so cannot have their package declaration altered. Even if these classes' sources were available it is likely that the developers of a system would be unwilling to take ownership of this code in order to improve its package structure. Considering only classes defined in source files is consistent with other efforts (Lakos 1996, Martin 1996b).

To get an idea of how the CRSS distribution relates to the structure of a PDG, consider again the PDGs shown in Figure 1. If we assume that each package has the same number of classes and that every class in a package depends on every other class in the same package then we get distributions like those in Figure 4. So, for example, if there are n classes in each package, then every class in a package in Figure 1(a) depends on every other class ($5n$ in total) in the system (a total of $5n$ classes), whereas only the classes in the middle package (a total of n) on the top row of Figure 1(c) depends on $3n$ other classes.

Figure 4 gives an indication of the kind of distributions we might see corresponding to PDGs with different characteristics, but what we need to know is, what does a given distribution tell us about the underlying PDG? The main contribution of this paper is that, if the CRSS distribution is such that there are 'many' classes with a 'large' CRSS value, then the current package structure for this system

cannot meet Lakos' model PDG (see Figure 2). Furthermore, and crucially, this situation indicates that the class relationships are such that there is *no* way to partition the classes to meet the Lakos model PDG, meaning that the only way to improve the package design is to change class relationships.

To see how certain distributions allow us to conclude that the package design is not as good as it could be, we need to be more specific about 'many' and 'large'. Rather than present the algebraic argument, we will give an indicative concrete example.

Suppose that we have a system with 1000 classes in it, and suppose we have decided that a package of 'manageable size' would have no more than 50 classes. If the package design for this system does not violate the manageable size principle, there must be at least 20 packages, and for this example we will assume there are exactly 20 packages of 50 classes each. The question is, how many stand-alone packages can there be, given a certain CRSS distribution.

The CRSS distribution we will consider is, 500 of the classes (\mathcal{L}) have CRSS values of 99 or fewer, and the other 500 classes (\mathcal{R}) have CRSS values of 600–699. The classes in \mathcal{L} could conceivably be partitioned into 10 (half) stand-alone packages, which is roughly consistent with the Lakos model PDG. So consider a class A in \mathcal{R} . It transitively depends on 600 or more other classes, and these 600 classes must be distributed over more than 12 packages (since 50 classes per package). At most 10 of those packages may involve only classes in \mathcal{L} , so the package containing A must depend on at least 2 other packages involving classes in \mathcal{R} . Since this is true for every class in \mathcal{R} , every package involving classes from \mathcal{R} must transitively depend on at least 2 other packages involving classes from \mathcal{R} . There can only be 10 such packages, so this is only possible if there is a cycle in the PDG, which means the any PDG for these classes cannot be in line with Lakos' tree model PDG.

The example given above may seem like an unlikely extreme cases, however, as we discuss in the next section, distributions similar to this are more common than one might expect. The advantage of the CRSS distribution is that it can be cheaply determined, and so quickly provides a reliable indication of the potential quality of the package design. Of particular advantage is that the information provided is independent of the actual package structure of the system we are measuring (see Section 7.1).

5 Results

5.1 A Software Corpus

We have developed a tool to compute CRSS from Java source files. We ran our tool over a corpus of Java software in order to determine the distribution of CRSS values in each of its programs. Programs selected for the corpus largely derive from the Purdue Benchmark Suite (PBS) used in an empirical study of type confinement (Grothoff, Palsberg & Vitek 2001). Programs in the PBS omitted from our corpus were those whose source code was not available. We have replaced these programs with others that we have previously used and whose source is freely available on the Internet.

The distributions of CRSS for each of the programs in our corpus are shown in the histogram of Figure 5. Being a histogram the horizontal axis shows the ranges of values for CRSS and the vertical axis shows the number of classes a given program that have that range of values for CRSS. The axis going 'into' the page shows each of the programs in the corpus, sorted by size, where this is measured in the number of top-level classes defined in the program's source. Again, since Figure 5 is a histogram, the heights of the bars for a given program sum to the number of (top-level source) classes in that program.

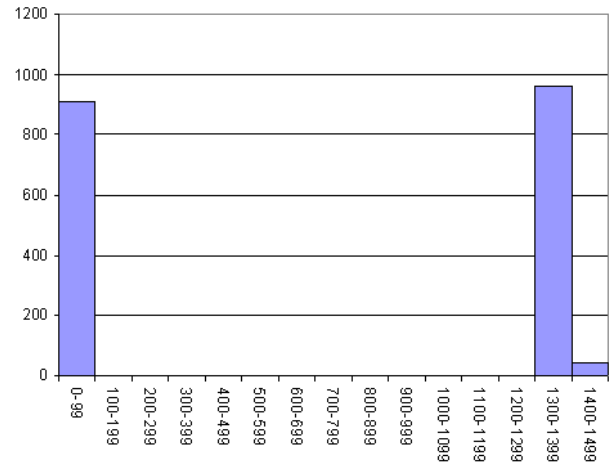


Figure 6: Azureus CRSS Distribution

Several of the programs in Figure 5 appear to have 'bad' CRSS distributions in that a large proportion of the classes in these systems have relatively high values for CRSS. We single out Azureus for further discussion because we were initially familiar with it from the perspective of an end-user and because there are space constraints on this paper. A histogram of CRSS values for Azureus is depicted in Figure 6 for the purposes of clarity.

5.2 Azureus

Azureus is peer-to-peer file-sharing client for the BitTorrent protocol. It was initially brought to our attention because it frequently appears on Sourceforge's title page in the top 10 lists for both downloads and development activity. We have used it and found that, at least from a user's perspective, it is a good piece of software because it is stable and easy to use.

The histogram of Figure 6 shows that there are approximately 1900 top-level classes defined in Azureus's source files (the sum of the heights of the bars). Of these 1900 classes about 900 have CRSS values of between 0 and 99. This means that each of these classes transitively depend on between 0 and 99 other classes. The remaining two bars combined show that about about 1000 classes depend on between 1300 and 1499 other classes. In fact, the transitive nature of CRSS means that none of the classes in the left-hand bar can depend on those in the right-hand bars. If a class from the left-hand bar depended on one in the right hand bars, it too would depend on 1300-1499 other classes so itself would have to be in the right-hand bars.

Table 1 shows a small selection of subsystems we have identified in Azureus many of which are not reflected in its current package structure. In column 2 of this table there is a representative or key class for each subsystem, or one that plays the role of Facade. The CRSS value given for the subsystem is computed from this class. As indicated in the 'CRSS' column of Table 1 each of the subsystems has a key class with a large CRSS value (ignore the 'CRSS-refact' column for now). This indicates that the subsystems depend on a great many other subsystems. Indeed we inspected the reachability sets of the classes in Table 1 and found that these key classes are actually mutually dependent. This means that the subsystems these key classes represent are also mutually dependent and that there must be a cycle among them.

Even without the knowledge that the classes of Table 1 are mutually dependent, the values in 'CRSS' column are still meaningful. For instance, it is hard to believe that a seemingly low-level subsystem like logging can depend on 1372 classes. If the maximum subsystem size of a logging subsystem's peers is 50 classes then it must transitively depends on *at least* 28 other subsystems. Contin-

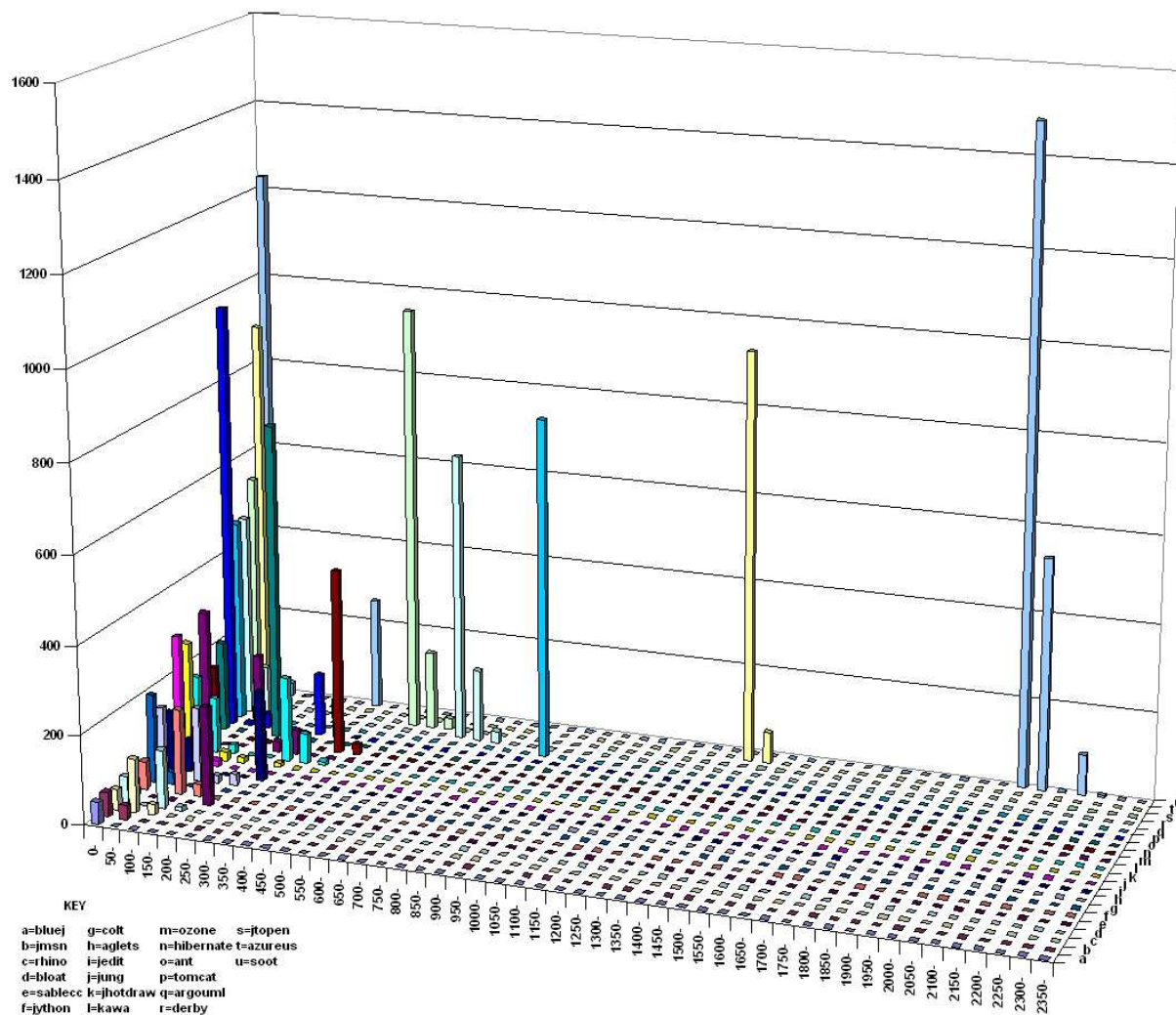


Figure 5: Software Corpus CRSS Distributions

Subsystem	Facade- or key-class	CRSS	CRSS-refact
debug	org.gudy.azureus2.core3.util.Debug	1372	16
threading	org.gudy.azureus2.core3.util.AEMonitor	1372	26
configuration	org.gudy.azureus2.core3.config.COConfigurationManager	1372	1360
internationalisation	org.gudy.azureus2.core3.internat.MessageText	1372	44
logging	org.gudy.azureus2.core3.logging.LGLogger	1372	12
torrent	org.gudy.azureus2.core3.torrent.TOTorrent	1372	32
time	org.gudy.azureus2.core3.util.SystemTime	1372	5
peer	org.gudy.azureus2.core3.peer.PEPeer	1372	171
disk-io	org.gudy.azureus2.core3.disk.DiskManager	1372	171
...

Table 1: Azureus subsystems

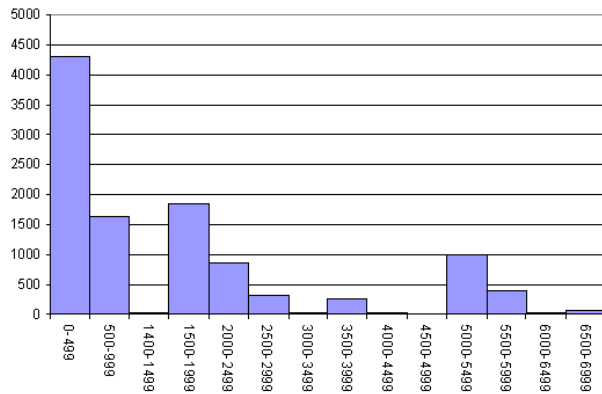


Figure 7: Eclipse CRSS Distribution

uing under the assumption that the maximum subsystem size is 50 classes then we can infer from Figure 6 that, irrespective of package structure, there are at least 20 subsystems represented in the right-hand bars and that these subsystems must each transitively depend on at least 26 other subsystems. The degree to which these subsystems is stand-alone is a far cry from Lakos's balanced binary tree reference model.

5.3 Eclipse

We also collected CRSS values for classes in the open-source IDE Eclipse, version 3.0.2 for Windows¹. The distribution of these CRSS values are shown in Figure 7. There are approximately 10700 top-level classes in Eclipse's source code. Figure 7 shows a decreasing trend in values for CRSS. Smaller values for CRSS appear to be more common than larger values. This is good because it means that dependencies between classes in Eclipse do not preclude it from having tree-like package structure. The right-most bar in Eclipse's CRSS distribution comprises only about 100 classes each of which transitively depend on 6500-6999 other classes. If the maximum package size at some level of abstraction is 500 classes it is feasible that only one package in the system transitively depends on 13 other packages. The taller the bar in the 6500-6999 the more packages that can potentially transitively depend on 13 other packages, thus the less stand-alone the packages that comprise Eclipse would be.

We do not present a table in the style of Table 1 for Eclipse because its size means that there are likely to be subsystems at many levels of abstraction. Instead we focus on two subsystems we have, in the past, wanted to lift from Eclipse for deployment in other programs. The first is Eclipse's Abstract Syntax Tree (AST) subsystem and the second is Eclipse's Resource Finder subsystem.

Eclipse's AST subsystem provides an Abstract Syntax Tree representation of a Java source file, or a set of Java source files. Other subsystems make use of this subsystem e.g. a Refactoring subsystem uses the AST for refactorings such as rename class, extract interface, override method. A Source Code Navigation subsystem uses this AST to perform operations such as goto declaration, open type hierarchy, find referring types. In essence the AST subsystem is a Java compiler front-end — it parses Java source code, does name bindings and produces an AST. The facade class for Eclipse's AST is `ASTParser`. We found that it has a CRSS value of 1572, which is we think is unusual because Sun's own Java compiler (for Java 5.0), which includes a back-end for writing ASTs to byte code comprises only 71 top-level classes.

There are several differences between Sun's Java compiler and Eclipse's AST subsystem that could cause a dif-

¹Eclipse is not shown in the corpus distribution because its size diminishes the heights of bars in the other programs too much.

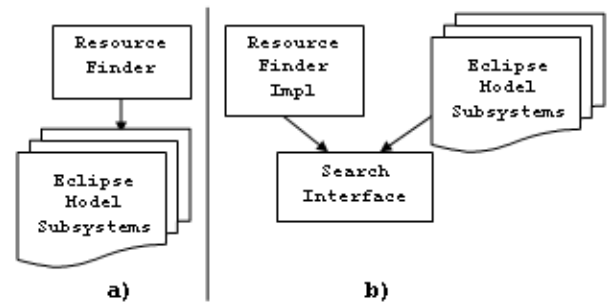


Figure 8: Resource Finder Subsystem

ference in CRSS values but none of which we think explain the magnitude of the difference. The differences are:

- Eclipse's AST subsystem relies on Eclipse project wrapper `IJavaProject` whereas Sun's gets external libraries, resources and source files off the classpath.
- Eclipse's AST subsystem allows the progress of the parsing and name binding to be monitored with `IProgressMonitor` although Sun's compiler also has a mode in which output messages could be interpreted to gauge the progress of the compilation.
- The AST node subclasses in Sun's compiler are public static inner classes whereas they are top-level classes in Eclipse's AST.

In any case none of these extra functions provided by Eclipse's subsystem should cause it to transitively depend on approximately 1500 more classes, especially since Sun's compiler provides the extra functionality of compiling to byte code. So we believe that Eclipse's AST subsystem could benefit from the type of refactoring depicted in Figure 3.

We have found the functionality provided by Eclipse's Resource Finder subsystem very useful when dealing with projects with many resource and source files. The Resource Finder dialog can be opened by pressing (`ctrl+shift+r`) in the IDE. This pops up a dialog that works by accepting a regular expression input and finding all files in open projects with filenames that match that regular expression. The facade class for the Resource Finder subsystem is `OpenResourceDialog` and has a CRSS value of 1945. Lifting 1945 other classes in order to reuse this Resource Finder subsystem is impractical considering our code inspections showed that the functionality provided by `OpenResourceDialog` is actually contained within only 5 classes. The problem is that `OpenResourceDialog` transitively depends on classes in Eclipse's model (c.f. view) (as shown in Figure 8(a)) when it should depend on some interfaces that are, in turn, passed into the model as shown in Figure 8(b).

6 Refactoring

6.1 Strategy

We have developed a refactoring strategy based on the *Dependency Inversion Principle (DIP)* (Martin 1996a) to reduce the number of classes in a system with large CRSS values. The strategy uses properties of the CDG to identify candidate classes for refactoring. The particular refactoring performed is *extract interface*, which may seem trivial, but we will see that eliminating the dependency on the implementation of the extracted interface is tricky. This trickiness occurs because at some point we must instantiate an interface with its implementation type — we refer

to this as the ‘problem of instantiation’, which is discussed below.

Performing the extract interface refactoring on a class reduces its clients’ CRSS values because the client classes no longer transitively depend on any types used in the extracted interface’s implementation. The effectiveness of the extract interface refactoring is dependent on many of the types referenced in the interface’s implementation not appearing in the signatures of the methods (and possibly fields) on the interface. In this way the CRSS value of the extracted interface is likely to be smaller than the value of its implementation. The transitive nature of reachability sets ensures that the clients of interface are likely to have smaller CRSS now than when they referenced what was effectively the interface’s implementation.

It follows that an effective way of reducing the CRSS values of many classes in a system is to extract interfaces from classes that are widely referenced and themselves have high values for CRSS. This is where the CDG comes in — a class is widely referenced if its CDG node has a large in-degree. Thus we identify candidate classes for the extract interface refactoring by sorting the list of classes in the system by in-degree then CRSS. While the extract interface refactoring is fairly simple to perform, dealing with client classes that need to instantiate the interface is not. In order to instantiate the interface we need to reference the interface’s implementation. If this is done through the use of a constructor call e.g. `Interface i = new Implementation();` we are in the same situation with respect to the client’s CRSS as before because the client still depends on the implementation. If this is done through reflection e.g. `Interface i = Class.newInstance("Implementation");` we still have a dependency on the implementation though our tool will not detect it and we have lost some of the type-safety provided by the language. Even if we use a factory class to return an instance of the implementation we still have a transitive dependency on the implementation through the call to the factory method that instantiates the class.

There are a number of ways of dealing with the problem of instantiation that are dependent on the way in which the interface is used. If the interface is instantiated only for a field in the client we can pass in the instantiation through the constructor:

```
public Client {
    private Interface i;
    public Client(Interface i) {
        this.i = i;
    }
}
```

In this way the class that instantiates the client also instantiates the interface’s implementation and passes it in through the client’s constructor. The client has no reference to the interface’s implementation. This technique is often referred to as *dependency injection*. Unfortunately it can result in more involved refactoring of the clients than simply textually replacing all references to the class’s name with its extracted interface’s name – sometimes extra parameters have to be added to the constructors and clients of the original clients need to be modified to instantiate the interface’s implementation.

We concentrate on performing the extract interface refactoring on candidate classes that are singletons because there is a means to instantiate these classes that puts little refactoring burden onto their clients. The ideal implementation of a singleton object through the use of a single static `getInstance`-type method and a private static field holding the instance. In reality we have found that singletons are implemented in a variety of ways (e.g., entirely using static methods and/or entirely using static fields). The solution we have for the problem of instantiation in the context of singletons involves the use of a *registry of singletons* (Gamma, Helm, Johnson & Vlissides 1995, p.130).

Class	In-degree	CRSS
...util.Debug	279	1372
...util.AEMonitor	168	1372
...config.COConfigurationManager	164	1372
...internat.MessageText	135	1372
...util.Constants	129	0
...download.DownloadManager	110	1372
...ui.tables.TableCell	110	7
...ui.tables.TableCellRefreshListener	108	7
...logging.LGLogger	107	1372
...bouncycastle.asn1.DERObject	103	3
...

Table 2: Candidates for Extract Interface Refactoring

We illustrate how the burden of refactoring on a singleton’s clients after the extract interface refactoring is performed on the singleton is reduced through the use of a registry of singletons. We illustrate this refactoring on the class A shown below, which gets split into `AIFace` and `AImpl`.

```
//this is the code pre-refactoring
public class Client {
    //inside some method
    A a = A.getInstance();
}

//this is the code post-refactoring
public class Client {
    //inside some method
    AIFace a = (AIFace)SingletonRegistry.get("A");
}

//this is a registry of singletons
public class SingletonRegistry {
    private Map m = new HashMap();
    public put(String key, Object value) {
        m.put(key, value);
    }
    public Object get(String key) {
        return m.get(key);
    }
}

//this line is needed somewhere near the entry
// point of the application to populate the
// registry with instances
singletonRegistry.put("A", new AImpl());
```

While we have used this refactoring only on singletons it can in fact be applied to non-singleton objects too, by also employing the *prototype* pattern (Gamma et al. 1995). In this way the *registry of singletons* becomes a *registry of prototypes*. In order to make an object a prototype for this purpose we must also add a method to its extracted interface (e.g., `newInstance`) that returns a new instance of the interface.

6.2 Results

We used our refactoring strategy on Azureus. Since Azureus has a variety of ways of implementing singletons e.g. the `getInstance`-method style, having all static fields, having all static methods we identified singletons manually from the list of candidates partially shown in Table 2.

The classes that we actually refactored were `LGLogger` (1), `COConfigurationManager` (2), `Debug` (3), `FileUtil` (4), `PlatformManager` (5), `MessageText` (6), `TorrentUtils` (7), `LocaleUtil` (8), `DisplayFormatters` (9), `Direct-ByteBufferPool` (10). The effect of these refactorings on the CRSS distribution are shown in Figure 9. The axis going ‘into’ the page has numbers that correspond to the extract interface operations on the listed classes. Each refactoring improved the distribution

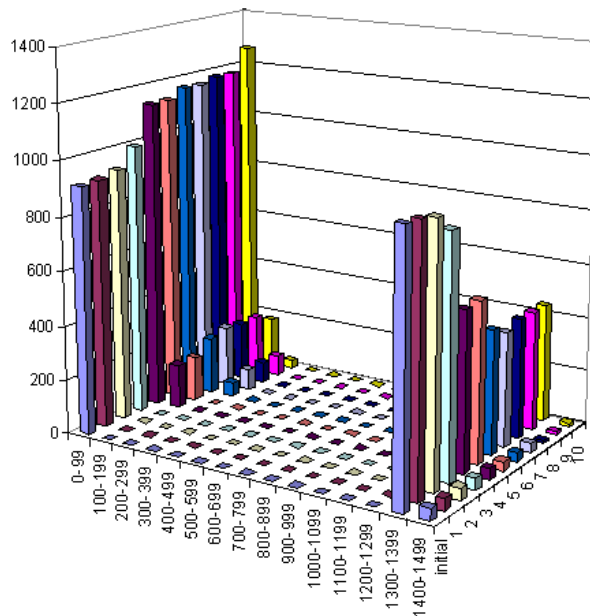


Figure 9: Refactoring Azureus

of CRSS as expected and after the 10th refactoring only 400 classes had CRSS values of 1300 or more and nearly 1300 classes now transitively depended on less than 100 other classes. The effects on the subsystems we identified earlier are shown in the ‘CRSS-refact’ column of Table 1. In the cases where the refactored class was the key class in the subsystem (i.e. Debug, COConfig-urationManager, MessageText and LGLogger) we show the CRSS value for the implementation, not the extracted interface since the former has the larger CRSS value. Indeed an inspection of the reachability sets of these subsystems now shows that they are no longer mutually dependent so could feasibly be arranged into packages without cycles in the PDG.

7 Related Work

The work in this paper extends a prior work (Melton & Tempero 2006b) and is also related to work we have done looking at dependency cycles among classes in Java software (Melton & Tempero 2006a). Here we review other work that has been done in metrics for package design. Hautus (Hautus 2002), Lakos (Lakos 1996) and Ducasse et al. (Ducasse, Lanza & Ponisio 2005) have each produced literature on this topic.

7.1 Hautus

The design principle stating a PDG should be a directed acyclic graph itself implies a simple metric. This metric classifies a given PDG as being either *cyclic* or *acyclic*. Unfortunately this metric is of little practical use, because we want to know the degree to which a cyclic PDG is cyclic. In this way we can estimate the amount of work required to make it acyclic, or determine if a refactoring has made it more or less cyclic. Hautus’s PASTA (Package Structure Analysis) metric aims to measure the degree of ‘cyclicness’ in a PDG.

The PASTA metric is defined for a given package as “the weight of the undesirable dependencies between the sub packages divided by the total weight of the dependencies between the sub packages” (Hautus 2002). The *weight* of a dependency is defined as “the number of references from one package to another”. Hautus does not make clear what constitutes a single reference — for instance references can be counted at the level of classes so

that a class can reference another at most once, or at the level of identifiers in the source code of a class so that a class can reference another multiple times. The *undesirable* dependencies are defined as a set of dependencies that when removed lead to an acyclic graph. Since there are multiple sets of dependencies that can be removed to lead to an acyclic graph the set is chosen such that it has the minimal weighted sum of references.

As Hautus’s metric is stated above it applies to a subgraph of a given PDG. The subgraph is chosen such that all its vertices are children of a given package in the package tree. In order to apply give the PASTA metric a single value for a whole program, rather than a single package, Hautus defines the PASTA metric for a whole program as “the weight of all desirable dependencies in all packages divided by the total weight of the dependencies in all packages”. This means that some references are counted multiple times since it is the underlying subpackage dependencies that gives a package its dependencies. Hautus states that this effect, of counting some references multiple times, is deliberate because it means that packages at a higher level of abstraction have a greater impact on the metric than those at a lower level of abstraction. Hautus then claims that it is more important to remove cycles between packages at a high-level of abstraction than cycles between packages at lower levels of abstraction.

Hautus’s metric differs from our CRSS metric in that it purports only to measure the ‘cyclicness’ of a PDG. This relates only to the single design principle that a PDG should be acyclic. We have argued that our metric is useful for indicating violations of other metrics, particularly *stand-alone* and *manageable size*.

Hautus has also produced a tool to collect his metric and support refactoring to eliminate cycles between packages. It appears that Hautus’s refactoring technique implicitly assumes that classes are correctly partitioned into packages and correspondingly that the way to remove cycles is to break dependencies between classes. This may not be a good assumption because repartitioning classes into a new package structure (especially with the support provided by Eclipse) is a far simpler operation than breaking dependencies between classes. Furthermore Martin claims that package design should be a bottom-up process whereby the class relationships dictate the formation of packages (Martin 1996b). Based on this statement it may be possible to use our CRSS metric as a starting point for determining how classes should be partitioned into packages.

7.2 Lakos

Lakos has identified several metrics for package design quality. The simplest of Lakos’s metrics is *Cumulative Component Dependency (CCD)*. CCD is the sums of the reachability set sizes for all the nodes in given PDG (Lakos 1996, p.187). Lakos also proposes an *average- and normalised-* version of this metric. We will discuss the average version. Average Component Dependency is ACD for a given PDG is CCD divided by the number of nodes in that graph.

Tall or cyclic PDGs will tend to have a higher value ACD than flatter, acyclic PDGs with stand-alone components (Lakos 1996, p.195). In this way ACD is useful for determining the degree to which a PDG follows the acyclic and flat package design principles. However, it does not take into account the size of a package so cannot be used to measure conformance to the manageable-size principle. It also deals with packages rather than classes so suffers from the same problem as Hautus’s.

7.3 Ducasse

Ducasse et al. introduce a number of metrics that could be used for measuring the package design quality, though

these metrics are discussed in the context of reverse engineering a system (Ducasse et al. 2005). In particular their paper concentrates on collecting metrics that can be used in visualisations of different types of dependencies between packages so the relationships between these packages can be more quickly and easily understood by a developer new to a system. Metrics from Ducasse et al. that could be useful for measuring package design quality are Number of Provider Packages (PP), Number of Client Packages (CC), Number of Class Clients (NCC) and Number of Classes in a Package (NCP). PP and CC correspond to the outdegree and indegree respectively of a package in a PDG. These could be used to indicate if a package was stand-alone or alternately excessively coupled to other packages. NCC could be used similarly – if many classes depend on a given package it may indicate that these classes/packages are excessively coupled and not stand-alone. NCP could be used to indicate if packages were of a manageable size.

8 Conclusions

Package design is believed to have an important effect on reusability and testability, as well as other quality attributes. It is therefore useful to know if the relationships between classes in a system preclude it from having packages that are stand-alone and of a manageable size. In this respect we have developed a simple metric, CRSS, that can be used to identify systems whose packages cannot be stand-alone and of a manageable size.

One distinguishing feature of our metric is that it is for *whole program* analysis — not just for individual elements of a program. Indeed it is the distribution of CRSS values for all the classes defined in the source files of a system that tells us about its best potential package structure.

We have presented empirical studies based on a number of open-source systems that identify distributions of CRSS that are indicative of package designs that cannot comprise packages that are stand-alone and of a manageable size. In order to improve the potential package structure of these systems we have shown how our CRSS metric can be used to identify good candidates for the *extract interface* refactoring. This refactoring can improve the relationships between classes in a system with respect to its potential for a good package structure.

References

- Bass, L., Clements, P. & Kazman, R. (1998), *Software Architecture in Practice*, Addison-Wesley Longman, Reading, MA.
- Binder, R. V. (1999), *Testing object-oriented systems: models, patterns, and tools*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- Booch, G. (1987), *Software components with Ada: Structures, tools, and subsystems*, Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA.
- Booch, G. (1991), *Object oriented design with applications*, Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA.
- Coad, P. & Yourdon, E. (1991), *Object-oriented analysis (2nd ed.)*, Yourdon Press, Upper Saddle River, NJ, USA.
- Ducasse, S., Lanza, M. & Ponisio, L. (2005), Butterflies: A visual approach to characterize packages, in 'Proceedings of 11th IEEE International Software Metrics Symposium (METRICS 2005)'.
- Gaffney Jr, J. E. & Cruickshank, R. D. (1992), A general economics model of software reuse, in 'ICSE '92: Proceedings of the 14th international conference on Software engineering', ACM Press, New York, NY, USA, pp. 327–337.
- Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995), *Design patterns: elements of reusable object-oriented software*, Addison-Wesley Longman Publishing Co. Inc., Boston, MA, USA.
- Grothoff, C., Palsberg, J. & Vitek, J. (2001), Encapsulating objects with confined types, in 'OOPSLA '01: Proceedings of the 16th ACM SIGPLAN conference on Object oriented programming, systems, languages, and applications', ACM Press, New York, NY, USA, pp. 241–255.
- Hautus, E. (2002), Improving Java software through package structure analysis, in 'The 6th IASTED International Conference Software Engineering and Applications'.
- IEE (1990), 'IEEE standard glossary of software engineering terminology'. IEEE Std 610.12-1990.
- Lakos, J. (1996), *Large-scale C++ software design*, Addison Wesley Longman Publishing Co. Inc., Redwood City, CA, USA.
- Martin, R. C. (1996a), 'The Dependency Inversion Principle', *C++ Report* 8(6), 61–66.
- Martin, R. C. (1996b), 'Granularity', *C++ Report* 8(10), 57–62.
- Melton, H. & Tempero, E. (2006a), An empirical study of cycles among classes in Java, in 'Tech. Report UoA-SE-2006-1', Department of Computer Science, University of Auckland.
- Melton, H. & Tempero, E. (2006b), Identifying refactoring opportunities by identifying dependency cycles, in 'CRPITS'48: Proceedings of the 48th conference on Computer science 2006', Australian Computer Society, Inc., pp. 35–41.
- Meyer, B. (1995), *Object success: a manager's guide to object orientation, its impact on the corporation, and its use for reengineering the software process*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA.
- Miller, G. A. (1956), 'The magical number seven, plus or minus two: Some limits on our capacity for processing information', *The Psychological Review* 63, 81–97.
URL: <http://www.well.com/user/smalin/miller.html>
- Shlaer, S. & Mellor, S. J. (1992), *Object lifecycles: modeling the world in states*, Yourdon Press, Upper Saddle River, NJ, USA.
- Szyperki, C. (1998), *Component software: beyond object-oriented programming*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.
- Thomas, D. & Hunt, A. (2002), 'Mock objects.', *IEEE Software* 19(3), 22–24.
- Wirfs-Brock, R., Wilkerson, B. & Wiener, L. (1990), *Designing object-oriented software*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Dynamic Measurement of Polymorphism

Kelvin H.T. Choi, Ewan Tempero

Department of Computer Science
University of Auckland
Auckland, New Zealand

kelvincht@hotmail.com; e.tempero@cs.auckland.ac.nz

Abstract

Measuring "reuse" and "reusability" is difficult because there are so many different facets to these concepts. Before we can effectively measure reuse and reusability, we must first be able to effectively measure these different facets. One such facet is the programming language constructs that are available. For example whether or not a language supports polymorphism is believed to affect how reusable a developer can make a code artifact. Effectively measuring polymorphism is a challenge because its behaviour is only observable at run-time. In this paper, we present a metric for polymorphism based on the dynamic behaviour of the code. We evaluate the usefulness of the metric through two case studies.

Keywords: software metrics, polymorphism, inheritance, dynamic profiling

1 Introduction

Measurement is considered important for many human endeavours. It is also often difficult to establish a form of measurement that allows management, especially prediction, to be done with confidence. Often we measure an attribute of interest by making a number of different measurements and combining them in a way that we believe (or hope) is representative of the original attribute. An example is "health". We have no way to measure health directly, and so we must make a number of measurements of other attributes, such as weight, blood pressure, cholesterol levels, and so on. So in order to measure health, we first have to be able to measure these attributes. So it is often the case that in order to measure what we want, we must identify all the relevant aspects and determine how to measure those first.

Managers of software engineers would like to be able to measure an engineer's productivity, but, as with health, there are many aspects that can affect productivity. Even measuring the individual aspects, were they clearly identified, is not easy. One such aspect is *software reuse*. Proponents of software reuse make many claims about how it affects productivity, but measuring that effect, and indeed even measuring how much reuse takes place, has proven difficult.

Copyright (c) 2007, Australian Computer Society, Inc. This paper appeared at the Thirtieth Australasian Computer Science Conference (ACSC2007), Ballarat, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 62. Gillian Dobbie, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

One confounding factor in measuring the benefits of reuse is that reusing different artifacts requires different amounts of effort. This means that we have the notion of *reusability*, the ease with which something can be reused, that must be taken into account when measuring reuse.

Polymorphism has been claimed to improve reusability, and so to fully measure productivity we may need some effective means to measure polymorphism. We also need to measure polymorphism in order to establish the veracity of the claims made about it with respect to reusability. It is the measurement of polymorphism that is the subject of this paper.

The problem with measuring polymorphism is that it is a concept whose behaviour is only seen at run-time. We can reason about what might be possible statically, but it is only dynamically that we can really see what is going on. In this paper, we present our attempt to measure polymorphism by means of software profiling. This is a technique that has been traditionally used for understanding other forms of dynamic behaviour, such as detecting "hot spots" for improving performance or determining the coverage of test cases. We apply similar techniques in an attempt to measure the polymorphism that takes place during the execution of an application.

The rest of the paper is organised as follows. The next section discusses the background and related work. Section 3 introduces the metric we propose for measuring polymorphism. Section 4 presents the methodology we used in our study of the metric and section 5 presents the case studies we carried out. We then discuss the results of our case studies and finally present our conclusions.

2 Background

2.1 Reuse, Reusability, and Polymorphism

The idea of developing software by re-using existing software has been around since the dawn of software engineering as a discipline (McClure 1997). Since that time, much research has been done to turn this idea into reality, of which we can only touch on here (see surveys such as (Krueger 1992; Mili et al. 1995) for more detail). For the most part this research has focused on planned or *systematic reuse*, that is, how organisations can, by using explicit processes and standards, get the most benefit from reuse. Early efforts in this regard examined the design, development, and organisational use of repositories of reusable assets (see (McClure 1997) for example). Later efforts considered domain engineering and domain analysis as promising avenues (see (Tracz et

al. 1993) for example), which led to software product lines (Clements and Northrop 2001).

Many authors have made a link between the reusability of code and the use of object-oriented concepts, for example (Meyer 1997). The common example is that of *inheritance*, and how it saves effort by supporting the easy use of an existing artifact without modification, that is, it aids **reuse**. Polymorphism is often lumped with inheritance, and so is accorded similar benefits with respect to reuse by implication. But in fact polymorphism has a more significant role for reuse than that. Consider the Java example below:

```
public interface XMLFormatter {
    public String asXML();
}
public void format(List<XMLFormatter> list) {
    for (XMLFormatter formatter: list) {
        System.out.println(formatter.asXML());
    }
}
```

The polymorphism that takes place here is in the call to the `asXML()` method on the `formatter` variable, since the contents of `list` could be any class that implements the `XMLFormatter` interface. Because polymorphism is being supported, the `format` method not only will work for any existing class that implements the `XMLFormatter` interface, *it will also work for any future such class*. It is instructive to consider what is being “reused” here. If it is the `XMLFormatter` interface, then not much (1 line) effort is being saved. Nor is it implementations of the interface, since they may have nothing in common. In fact, what is being reused here is the `format` method, since anyone needing this functionality can just use this method. If we didn’t have polymorphism, this would not be possible, so what polymorphism makes it easier for us to write code that can be used in multiple situations, that is, it supports **reusability**.

2.2 Measuring Reuse and Reusability

There have been a number of efforts in measuring reuse. Most of the research into measurement relating to reuse has focussed on the economics of reuse. Software reuse is only beneficial if the return on investment (ROI) (Kain 1994) is positive. There are costs in managing the reuse process. This includes the cost of maintaining the reuse libraries, the cost to modify and repackaging reusable artifacts from the existing asset, and the cost of searching, identifying, evaluating, selecting and integrating the potential artifact.

Various economics metrics have been proposed to measure the cost-benefit ratio of reuse. Gaffney and Durek’s reuse economic metric (Gaffney and Durek 1989) is useful to measure the cost of reuse and its break-even point. Reuse is economically break-even when the cost of the reuse equals the reuse benefit. Balda and Gustafson’s COCOMO-based reuse model (Balda and Gustafson 1990) can estimate the total time it would take to build software with reuse. Barnes and Bollinger’s cost-benefit analysis (Barnes and Bollinger 1991), on the other hand is focused on the quality benefits.

Henderson-Sellers’s cost-benefit analysis (Henderson-Sellers 1993) is focused on calculating the ROI from reuse activities. Malan’s cost-benefit analysis with NPV (Malan and Wentzel 1993) can be used to determine the financial cost saving from reuse and their fixed costs. Poulin and Caruso’s reuse metrics and ROI (Poulin and Caruso 1993) can be used to calculate and differentiate Project-level ROI and Corporate-level ROI. Finally the U.S. Defense information system agency (DISA)’s reuse metrics and ROI (DISA/JIEO/CIM 1993) is focused on measuring cost avoidance.

The Gaffney and Durek’s reuse economic metric is representative of many of these. It is

$$C = (b + E/n - 1)R + 1$$

Where

C = relative cost of software development (generally C will be less than 1, less is better)

R = the proportion of reused code in the project ($0 \leq R \leq 1$)

b = the cost, relative to new code, of reusing existing code in this project. (For example searching, adaptation and integration cost)

E = the cost, relative to new code, of developing a component for reuse. (For example the cost to make code reusable)

n = the number of expected reuses

This metric requires the cost of creating the reused artifacts (E) as input, whereas our interest is ultimately in being able to measure this cost.

There are different interpretations about what to measure as reuse. Poulin and Kain (Kain 1994; Poulin 1997) only count external reuse (reuse across project boundaries) as reuse. For example, they do not count copy and paste, or programming languages features such as sub-classing and polymorphism, as reuse. Also they do not count modified component and Commercial Off-the-Shelf Software (COTS) as reuse. They do not give additional credit for how many times a component is reused. On the other hand, both Henderson-Sellers (Henderson-Sellers 1993) and Frakes (Frakes and Carol 1994) have another perspective about reuse. They view sub-classing via inheritance as reuse, count internal reuse as reuse, and give credit for how many times components are reused. Poulin and Kain emphasise ‘industrial benefits’ of reuse, whereas Henderson-Sellers and Frakes emphasise the academic definition of reuse.

Two commonly cited reuse metrics are those by Banker et al (Banker et al. 1994) and Frakes (Frakes and Carol 1994).

Reuse Percentage

Banker et al. developed a metric in a repository-based CASE environment named High Productivity Systems (HPS). Their metric measures the level of reuse using reuse percentage:

$$\text{Reuse Percentage} = (1 - \text{New objects built} / \text{Total objects used}) \cdot 100\%$$

For example suppose a project uses 400 objects, however the developers in this project only build 100 objects. In this case the reuse percentage would be 75%. This metric counts multiple invocations of the same object as multiple instances of reuse. For example 200 uses may come from using 50 objects 4 times each. Also this metric measures both internal and external reuse within the project. For example from the 400 uses, 200 uses may come from internal reuse and the other 200 may come from external reuse.

This metric does not precisely define object granularity. This does not obey the fourth property (Devanbu et al. 1996) discussed by Devanbu et al., which states that reuse benefit measures should be sensitive to the cost of the object being reused. In the above metric, reusing a small object may yield the same reuse benefit as a large object.

Reuse Level Metric

Frakes (Frakes and Carol 1994) introduced the Reuse Level Metric. This metric uses threshold levels to filter out the items that are not reused often enough. For example if the threshold level is 3, an item would have to be called at least 4 times in order to be counted as reuse. Also this metric differentiates between internal reuse level and external reuse level.

Internal reuse level = IU / T

External reuse level = EU / T

Total reuse level = Internal reuse level + External reuse level

Where

IU = number of external items reused, that is more than the external threshold level.

EU = number of internal items reused, that is more than the internal threshold level.

T = the total numbers of items in the system, both external and internal

Each reused item (such as IU and EU) only has a value 0 or 1. If the item is reused more than the threshold level, it will always be 1, otherwise it will be 0. This means that this metric does not take into account how many times an item is used.

Note that the Reuse Level Metric uses items instead of lines of code (LOC), since each item would be different and some items would be very large and some items would be very small. Another version of this metric assigns a weight to each item depending on the item's size. It also allows different threshold levels for internal and external reuse. This means that in order to compare results between projects, their threshold levels must be same. Otherwise, projects with smaller threshold values will appear better.

3 Measuring Polymorphism

The problem with measuring polymorphism is that exactly what happens can vary from execution to execution of an application. In the earlier example, the implementation of the `format` method that is called can

potentially change every time through the loop. In order to better understand what benefit we are getting through the use of polymorphism, we would like to characterise what actually happens.

3.1 Polymorphic behaviour index

Consider the following example, which provides a simpler view of polymorphism in action than our earlier example:

```
List list; //general type declaration
If (external condition)
    list = new ArrayList();
else
    list = new LinkedList();
list.add(...);
```

In the example, whether the call to the `add` method on the `list` variable causes the `ArrayList` implementation or the `LinkedList` implementation to be called depends on the external condition. For any given condition, we can not always tell whether or not that condition is ever satisfied, so while it looks like polymorphism will take place in our example, if the condition is never true (or false) then in fact no polymorphism will happen. In this particular example, because the declared type of `list` is an interface type, any method invoked on it will involve a polymorphic call, but that will not always be the case.

We can use dynamic analysis to get better information. At runtime, when the `add` method is called, the actual class of the `list` variable is known, and so the actual implementation being used can be determined. Whether or not polymorphism has actually occurred depends on whether the type of the object on which the method call is dispatched agrees with the declared type of the variable.

This gives the basis for our metric. We can determine whether or not a method call is polymorphic as described above. This gives us a simple metric that relates to the amount of polymorphism that takes place:

Polymorphic behaviour index = $P / \text{Total dispatches}$

Where

Total dispatches = $(P + NP)$

P = Unique polymorphic dispatches executed

NP = Unique non-polymorphic dispatches executed

The **Declared Interface** is the interface of the variable that is declared in the source code. In the example: `List list = new ArrayList();` the `list` variable will be declared using the `List` interface.

The **Dispatched Class** is the class that the method is invoked on. In the example, if the external condition is true, then the dispatched class will be `ArrayList`.

Conforms and *implements* are relations between a Class and an Interface, and they are different. Any class that implements an interface must also conform to that interface. However a class that conforms to an interface might not directly implement it. The implementation of an interface method may come from its parent class. The

method called from an interface is dispatched to the implementation of that interface.

When an object's method is called, the actual method that is executed is the deepest inherited method that has an implementation. Consider this example:

```
Class Parent{
    void method3(){};
}
class Child extends Parent{
    Child c = new Child();
    c.method3();
}
```

The `c.method3()` will be dispatched to `parent.method3()` because it is the deepest inherited method that has an implementation.

Polymorphic dispatch is detected when the Declared Interface and the Dispatched Class is different.

Non-polymorphic dispatch is detected when the Declared Interface and the Dispatched Class is the same.

For example:

```
interface I{
    m();
}
class Parent implements I{
    m(){ //a implementation of method m()
        ...; //do something
    }
    m2(){...; //do something }
}
class Child extends Parent{
    m2(){
        super();
        ...; //do something more specific
    }
}
```

When:

```
Parent obj1 = new Parent();
obj1.m();
```

The `obj1` variable points to a `Parent` Object. The `Parent` class has a method `m()` implementation so `Parent.m()` is dispatched. **The Dispatched Class of method `m()` in this case is `Parent`.** This dispatch is **non-polymorphic** because the Declared Interface is the same as the Dispatched Class; `Parent` in this case.

```
Child obj2 = new Child();
obj2.m();
```

Although `obj2` points to an Object of type `Child`, the method that is dispatched is `Parent.m()`. Since `Child` does not have a method `m()` implementation, the *deepest* inherited method that has such implementation is `Parent.m()`. **The Dispatched Class in this case is `Parent`.** This dispatch is **polymorphic** because the Declared Interface is `Child`, whereas the Dispatched Class is `Parent`.

```
I obj3 = new Parent();
obj3.m();
```

In this case, although `Parent.m()` will be dispatched. However the Declared Interface is `I` and the Dispatched

Class is `Parent`. This dispatch is **polymorphic** because Declared Interface = `I`, Dispatched Class = `Parent`.

Inherited Method Call using inherited class

```
Parent obj4 = new Child();
obj4.m();
```

In this case, the dispatched method is `Parent.m()` and the Declared Interface of `obj4` is `Parent`. So the dispatch is **Non-polymorphic** because Declared Interface = `Parent`, Dispatched Class = `Parent`.

```
Parent obj5 = new Child();
obj5.m2();
```

In this case `obj5` uses the inherited method `m2()` from `Child`. `Child.m2()` overrides the `Parent's m2()` and this inheritance reuses the effort from the parent's `m2()` method. The dispatched method is `Child.m2()` because it is the deepest method that has an implementation. So the dispatch is **polymorphic** because Declared Interface = `Parent`, Dispatched Class = `Child`.

Consider the following example with loop:

```
static void m1(ArrayList arrayList){
    for (int i=0;i<10;i++){
        arrayList.add(.); //this is non-poly..
        //Declared Interface = ArrayList,
        //Dispatched Class = ArrayList
    }
}
static void m2(List list){
    for (int i=0;i<10;i++){
        list.add(...); //this is polymorphic
        //Declared Interface = List,
        //Dispatched Class= ArrayList
    }
}
```

In the above example, `m2()` supports polymorphism and it is more reusable than `m1()`, because the input parameter of `m2()` uses an interface `List` instead of a strict `ArrayList`. This means that other people using `m2()` can use all kinds of `List` such as `LinkedList`, `Vector` and `ArrayList`. However people using `m1()` can only use `ArrayList`.

Now consider this case

```
//it is forced to use ArrayList
m1(new ArrayList());
```

When `m1()` is called, the code inside `m1()` will be executed. Within the `m1()` loop, `arrayList.add()` will be called 10 times.

The Declared Interface of `arrayList` is `ArrayList`. Since `arrayList` points to an object of `ArrayList`, the dispatched method is `ArrayList.add()`. In this case the declared interface and dispatched class are same so this will be in non-polymorphic. Also although there are 10 calls, they all occur at one call site due to a single call to `m1()`, so they will be counted as one *unique* dispatch.

Since there are no polymorphic dispatches but there is one non-polymorphic dispatch, in this case:

Polymorphic behaviour index = $0 / (0+1) = 0\%$

Consider another case:

```
m1(new ArrayList()); //non-polymorphic
m2(new LinkedList()); //polymorphic
m2(new Vector()); //polymorphic
m2(new ArrayList()); //polymorphic
```

In this example, `m2()` is called three times and `m1()` is called once. `m2()` is polymorphic and it takes all types of lists. The `ArrayList.add()` method within the `m1()` loop will be executed 10 times and there will be 1 unique non-polymorphic dispatch. Also `list.add()` method within the `m2()` loop will be executed. The `list.add()` method within `m2()` has a Declared Interface of `List`.

- The `list.add()` will be executed with `LinkedList` 10 times. Dispatched Class = `LinkedList.add()`. The method calls will be count as one unique polymorphic dispatch.
- The `list.add()` will be executed with `Vector` 10 times. Dispatched Class = `Vector.add()`. The method calls will be count as one unique polymorphic dispatch.
- The `list.add()` will be executed with `ArrayList` 10 times. Dispatched Class = `ArrayList.add()`. The method calls will be count as one unique polymorphic dispatch.

The total number of unique polymorphic dispatches is 3 and the total number of unique non-polymorphic dispatches is 1.

Since there are 3 polymorphic dispatches and one non-polymorphic dispatch, in this case:

Polymorphic behaviour index = $3 / (3+1) = 75\%$

3.2 External and internal reuse

Software reuse can be classified as internal reuse or external reuse. External reuse occurs when calls are made to code supplied from a source external to the project, what is often referred to as an “API”, whereas internal reuse is when calls are made to code that has been already written for the application. Some people believe external reuse is most beneficial and so internal reuse should not be counted. However internal reuse is also important. For example developers reusing their own methods are more productive than creating another new method that does the same thing. For this reason, we measure both internal and external reuse.

Internal methods are defined as methods that are created by the application developers. They also include methods implementing the API interface or custom code extending the API.

External methods are defined as methods that are supplied from external source and which developers cannot change. These include API and external libraries.

An *external method call* happens when an *internal method* calls an *external method*. This excludes an *external method* calling another *external method*. The

reason for this is because the benefit of reusing the external method directly called by an internal method is equivalent to reusing all of the methods that that external method reuses. Therefore it is not required to re-count the inter-external method reuse.

An *internal method call* happens when any *method* calls another *internal method*. In most cases it will be internal methods calling internal methods. However sometimes internal methods are used as handlers and are passed into the API and the API will call them. In that case they are still counted as internal method calls because such external to internal method calls is a consequence of the developer’s decision just as with other internal to internal method calls.

Consider the following example:

```
class Example extends java.lang.Thread{
    void methodA(){
        methodB();
    }
    void methodB(String s){
        java.lang.System.out.println("abc");
    }
    /*this method implements the run() method required in
    thread. When Thread.start() is called, this method
    will be called by the JVM scheduler.*/
    public void run(){
        methodA();
    }
    public static void main(...){
        this.start(); //this class extends Thread.
        // This will call Thread.start()
    }
}
```

When this `Example` class is executed, the `main()` method will be called by the *Java Launcher*. Since `Example` extends `Thread`, calling the `start()` method will put this `Thread` into the *Java Scheduler*. Eventually the *Java Scheduler* will call the `run()` method and the `run()` method will call `methodA()`, which will, in turn, call `methodB()`. `methodB()` will then call `System.out.println()`.

There are many types of method calls in this example:

The internal method calls consist of:

- `methodA()` calling `methodB()`, both are *internal methods*
- *Java Scheduler* calling `run()`, notice the *Java Scheduler* is an API and it is not in the code. It is *external to internal* method call and it is counted as internal method calls.
- `run()` calling `methodA()`, `run()` conforms to the `Thread.run()` interface, but it is an *internal* method because it is created by the application developer. `methodA()` is an *internal* method. It is *internal to internal* method call.
- *Java Launcher* calling `main()`, notice the *Java Launcher* is an API and it is not in the code. It is *external to internal* method call.

The external method calls consist of:

- `methodB()` calling `System.out.println()`, internal to external method calls
- `main()` calling `start()`, `main()` is an internal

method implemented by the developer. The `start()` method from `Thread`. Although `Example.start()` method is called and `Example` is an *internal* class, `Example` does not have an implementation of `start()`. It inherits the `Thread.start()` implementation. So the `Example.start()` method is an *external* method. `main()` is *internal* and `start()` is *external* so it is *internal to external* method call.

There are many *external to external* method calls, for example when `System.out.println()` is called, many inter-API calls will be made such as the `StringBuffer` calling `PrintWriter`, etc. However we will not classify them as either internal or external method calls.

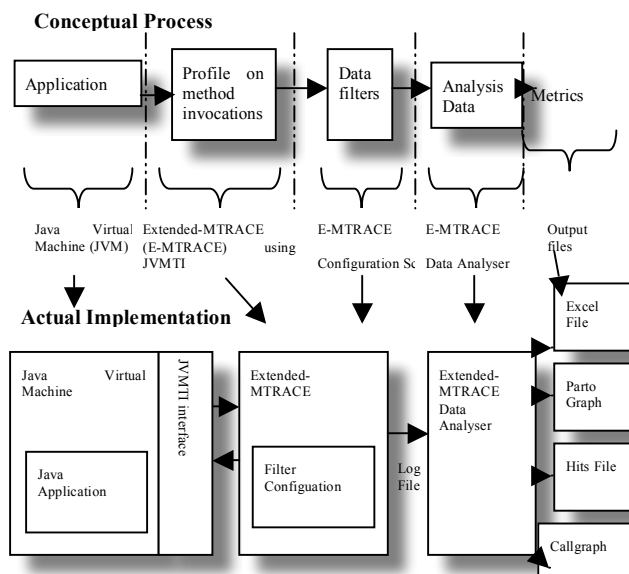


Figure 1 the Process for Dynamic Metrics.

4 Methodology

Our methodology for evaluating our metric is to compare two software applications from the same domain both in terms of our metric, and by manually examining the source code to determine their relative reuse and reusability performance.

Computing the metric requires instrumenting the compiled version of each application. Each application is then executed via a script that provides the same workload to each to produce a profile of their execution.

This will result in runtime data such as: method calls, call sites, Dispatched Class and so on. A filter is needed to remove the unnecessary results such as system method calls made by the JVM. Then the filtered data is analysed to identify polymorphic and non-polymorphic dispatches, from which the metric value is computed.

The tool we have developed to do all this is called E-MTRACE (Choi 2006). E-MTRACE uses the JVM Tool Interface (JVMTI), which provides a means to inspect and control the execution of programs running in the Java Virtual Machine. E-MTRACE extends the JVMTI demo MTRACE. E-MTRACE first inserts bytecodes into methods by using a File Hook. A File Hook interrupts the JVM when the JVM wants to load the java .class file into the Heap. The Hook interprets the original .class file and inserts profiling instrumentation code before any method that is called. This allows instrumentation code to be inserted during the runtime. When a method is called, the instrumentation code is executed. The instrumentation code uses the program stack to identify the Dispatched Class. It then uses the Local Variable Table to trace the Declared Interface. The details is discussed in the Choi's Thesis (Choi 2006). Once the Declared Interfaces and Dispatched Classes are identified, they will be processed using the E-MTRACE Analyser tool, which produces the list of polymorphic dispatches and non-polymorphic dispatches. An example is shown in figure 2.

5 Case Study

We performed a case study to evaluate the polymorphic behaviour index metric. The case study consisted of comparing two Search Engines – Egothor (Galambos 2006) and Lucene (Bialecki et al. 2005) and two FTP servers – Dano (Paregos 2002) and Jupiter (Filion 2002) using the polymorphic behaviour index.

5.1 Search Engines

The two search engines are Egothor (Galambos 2006) and Lucene (Bialecki et al. 2005). Egothor is a fully featured search package and it can be configured as a full-text search engine, meta searcher and crawler robot. It uses a pipe-line architecture. It also uses an extendable filter architecture. The search algorithm can automatically find the cheapest search-plan (the order of search-operations executed) using weighting. The plan can be automatically compiled and executed. Search operations are dispatched into separate thread allowing it to take advantage of a multiple CPU machine.

	A	B	C	D	E	F	G	H
449	DeclaredType	DispatchedClass	ToMethod	FromMethod	Variable			
450	java/io/DataInput	java/io/DataInputStream	java/io/DataInputStream.readU	org/egothor/store/util/	org/egothor/store/util/Compress.readPack			
451	java/io/DataInput	java/io/DataInputStream	java/io/DataInputStream.readU	org/egothor/store/util/	org/egothor/store/util/Compress.readPack			
452	java/io/DataInput	java/io/RandomAccessFile	java/io/RandomAccessFile.read	org/egothor/store/util/	org/egothor/store/util/Compress.readPack			
453	java/io/DataInput	java/io/RandomAccessFile	java/io/RandomAccessFile.read	org/egothor/store/util/	org/egothor/store/util/Compress.readPack			
454	java/io/DataInputStream	java/io/FilterInputStream	java/io/FilterInputStream.close	org/egothor/db/disc/Dorg/egothor/db/disc/DiscIndexData.getM				
455	java/io/InputStream	java/io/BufferedInputStream	java/io/BufferedInputStream.cl	org/egothor/store/disc/	org/egothor/store/disc/ThickBarrel.setLoc			
456	java/io/InputStream	java/io/FileInputStream	java/io/FileInputStream.close	org/egothor/db/disc/Dorg/egothor/db/disc/DiscKeyIndexData.i				
457	java/lang/Object	java/lang/String	java/lang/String.toString	org/egothor/db/disc/Dorg/egothor/db/disc/DiscKeyIndexData.i				
458	java/lang/Object	java/lang/String	java/lang/String.charAt	org/egothor/db/disc/Dorg/egothor/db/disc/DiscKeyIndexData.i				
459	java/lang/Object	java/lang/String	java/lang/String.length	org/egothor/db/disc/Dorg/egothor/db/disc/DiscKeyIndexData.v				
460	java/lang/Object	java/lang/String	java/lang/String.toString	org/egothor/db/disc/Dorg/egothor/db/disc/DiscKeyIndexData.v				
461	java/util/Enumeration	java/util/Vector.\$1	java/util/Vector.\$1.hasMoreEle	org/egothor/query/QCorg/egothor/query/QGroup.addTermsO.e				
462	java/util/Enumeration	java/util/Vector.\$1	java/util/Vector.\$1.nextElement	org/egothor/query/QCorg/egothor/query/QGroup.addTermsO.e				
463	java/util/Enumeration	java/util/Vector.\$1	java/util/Vector.\$1.hasMoreEle	org/egothor/query/QCorg/egothor/query/QGroup.applyCWI0.e				
464	java/util/Enumeration	java/util/Vector.\$1	java/util/Vector.\$1.nextElement	org/egothor/query/QCorg/egothor/query/QGroup.applyCWI0.e				
465	java/util/Enumeration	java/util/Vector.\$1	java/util/Vector.\$1.hasMoreEle	org/egothor/query/QCorg/egothor/query/QGroup.attachO.e				
466	java/util/Enumeration	java/util/Vector.\$1	java/util/Vector.\$1.nextElement	org/egothor/query/QCorg/egothor/query/QGroup.attachO.e				
467	java/util/Enumeration	java/util/Vector.\$1	java/util/Vector.\$1.hasMoreEle	org/egothor/query/QCorg/egothor/query/QGroup.explainO.e				
468	java/util/Enumeration	java/util/Vector.\$1	java/util/Vector.\$1.nextElement	org/egothor/query/QCorg/egothor/query/QGroup.explainO.e				
469	java/util/Enumeration	java/util/Vector.\$1	java/util/Vector.\$1.hasMoreEle	org/egothor/query/QCorg/egothor/query/QGroup.setModelO.e				
470	java/util/Enumeration	java/util/Vector.\$1	java/util/Vector.\$1.hasMoreEle	org/egothor/query/QCorg/egothor/query/QGroup.toStringO.e				
471	java/util/Enumeration	java/util/Vector.\$1	java/util/Vector.\$1.nextElement	org/egothor/query/QCorg/egothor/query/QGroup.toStringO.e				
472	java/util/Iterator	java/util/HashMap\$HashIterator	java/util/HashMap\$HashIterat	org/egothor/db/disc/Dorg/egothor/db/disc/DiscKeyIndexData.i				
473	java/util/Iterator	java/util/HashMap\$HashIterator	java/util/HashMap\$HashIterat	org/egothor/db/disc/Dorg/egothor/db/disc/DiscKeyIndexData.v				
474	java/util/Iterator	java/util/HashMap\$KeyIterator	java/util/HashMap\$KeyIterator	org/egothor/db/disc/Dorg/egothor/db/disc/DiscKeyIndexData.v				

Figure 2 Polymorphic dispatches output data

Lucene provides search engine functionality through a simple API. It is intended to be a scalable, fast, cross platform system providing full-text search. The design uses recursion method calls and many handlers.

Egothor provides more features than Lucene, but both provide full-text searching, so that functionality is profiled and compared. Egothor and Lucene use different design architectures to implement their core engine and so it is a useful comparison. In order to compare the two search engines, their mutual inclusive functionalities are used and these include:

- ☐ Index search
- ☐ Create index
- ☐ Search syntax operators such as AND OR NOT
- ☐ Dynamically update index

Searcher	Reuse type	Polymorphic behaviour index (more the better)	Polymorphic dispatches	Non polymorphic dispatches
Egothor	External	64%	45	25
Egothor	Internal	40%	46	70
Lucene	External	20%	5	19
Lucene	Internal	29%	52	127

Table 1 Searchers Polymorphic Dispatches

The results are shown in Table 1. Egothor (both externally and internally) has a higher polymorphic behaviour index than Lucene. Also Lucene has significantly more internal dispatches than external dispatches.

When inspecting the Lucene code and design, the internal design of Lucene confirms the internal reuse and also significant use of inheritance. Therefore it has a lot of polymorphic dispatches – 52. For example there are many polymorphic types of keywords (Terms) and Keyword-Indexes.

However the inheritances trees are disjointed and the trees are shallow. For example there are many types of Index, there are many types of Terms but Term and Index do not inherit from each other. That is why it has many polymorphic dispatches – 52 but there is also a lot of non-polymorphic dispatches -127. It only has 29% of internal polymorphic behaviour index.

Externally, Lucene does not have many dispatches (either polymorphic or non-polymorphic) at all. Egothor on other hand is very good on dispatching external API and it even inherit on the external API (64% external polymorphic). For example its Cache extends HashMap, and its in-house developed QueryHit extends the standard Java Iterator.

The Egothor internal polymorphic behaviour usually comes from different Query Types extending the Query Interface. For example QueryTerm and QueryGroup extends Query. Also Egothor's internal design is a procedural, linear pipeline and uses filters. For example it is structured in steps:

- ☐ Step 1: Find cheapest search-plan
- ☐ Step 2: compile search-plan
- ☐ Step 3: execute search-plan.

Each step makes many calls to the external API and reuses the internal plan elements such as Nodes and the Edges (filters). Also filters are polymorphic and they can be extended. For example BuildHTML extends Edge. BuildHTML takes the input of data tokens from previous Node and convert it into HTML documents.

However there are places where inheritance could have been used, but was not. For example there are three BuildHTML classes. They are BuildHTML1, BuildHTML2 and BuildHTML3. All of them are similar because they build different format of the HTML document. Unfortunately all of them only extend edge. It is not enough and it can be done better, for example:

Egothor way

```
BuildHTML1, BuildHTML2, BuildHTML3 extends Edge
```

Better way

```
AbstractBuildHTML extends Edge
BuildHTML1, BuildHTML2, BuildHTML3 extends
AbstractBuildHTML
```

Assuming the commonality between the BuildHTML classes exist, the common functions of the BuildHTML classes should be extracted and put into the AbstractBuildHTML, so when a new BuildHTML4 is created less effort is needed because it can reuse the code in the AbstractBuildHTML. If such a design were used, then we would probably see a higher polymorphic behaviour index. This could be one reason why the internal polymorphic behaviour of the Egothor (40%) is less than the external polymorphic behaviour (64%).

Overall, Egothor has a higher percentage of polymorphic behaviour than Lucene.

5.2 Ftp Servers

The two FTP servers are Dano (Paregos 2002) and Jupiter (Filion 2002). The implementation is quite different between the two. Jupiter is based on a simple design that uses as few classes as possible. Dano is based on a highly extendable design that has many little classes and each of them is responsible for different functions. In order to compare two FTP servers, their mutual inclusive functionalities are used and these include:

- ☐ Browsing a folder
- ☐ Upload a file
- ☐ Download a file
- ☐ Move/Copy/Remove file

FTP Server	Reuse type	Polymorphic behaviour index (more the better)	Polymorphic dispatches	Non polymorphic dispatches
Dano	External	24%	7	22
Dano	Internal	63%	59	34
Jupiter	External	12%	7	52
Jupiter	Internal	0%	0	22

Table 2 FTP Servers polymorphic dispatches

The results are shown in table 2. It shows that Dano has significantly higher external and internal (24%, 63%) polymorphic behaviour index than Jupiter (12%, 0%). It is interesting that Jupiter internally has zero polymorphic dispatches.

Dano has higher internal polymorphic behaviour index (63%) than externally (24%), whereas Jupiter has lower internal polymorphic behaviour index (0%) than external polymorphic index (12%).

Comparing external polymorphic behaviour

Dano FTP extends its FTPConnection on a Java Thread object. Jupiter on the other-hand does the same thing, but with three classes: ConnectionThread, FileTransferThread and PasvListeningThread. It appears Jupiter is more multi-threaded than Dano.

Dano indirectly made some external polymorphic dispatches by using the polymorphic Factory API, such as:

```
InetAddress ia = InetAddress.getLocalHost();
...//Call ia.anyMethod()
```

The variable `ia` is declared as `InetAddress`, but using the `InetAddress.getHost()` method returns an `Inet4Address` object. Therefore variable `ia` is declared as `InetAddress` but the actual class is `Inet4Address`. It means using a polymorphic factory API will improve the polymorphic behaviour index.

Both of the FTP server applications scored the same number of internal polymorphic dispatches (7), however Jupiter has more non-polymorphic external dispatches (52) than Dano (22). Therefore Jupiter has proportionally fewer polymorphic dispatches (24%) than Dano (12%).

Comparing internal polymorphic behaviour

Internally it is a different story. Dano designs for polymorphism while Jupiter does nothing at all. Dano has 63% internal polymorphic behaviour index but Jupiter has 0%. This can be explained by the architecture:

Jupiter only has a small number of Java files. However each of them is very big and does many things. For example, Jupiter puts all the FTP-commands handling logic into one class called `ProtocolInterpreter`. It is a huge class and it is very difficult to understand. Also many things are hard-coded.

If the FTP protocol is changed, for example command "SYST" is renamed "SYSE", the whole of `ProtocolInterpreter` needs to be inspected and updated.

On the other-hand, Dano uses inheritance a lot internally. For example each different FTP command has its own class and all of them extend `FTPCommand`.

```
public interface FTPCommand {
    //Sets the FTPUserContext before the //command
    executes.
    Public void setUserContext( FTPUserContext uc );
    //Sets the FTPCommand argument string.
    public void setArgument( String arg );
    //Executes the command.
    public void runCommand( ) throws
        FTPException;
    //Aborts the command.
    public void abortCommand( );
}
```

Inside Dano, there are 28 Command classes that extend the `FTPCommand` interface. For example, within `SYST`:

```
public void runCommand( ) {
    pi.sendResponse( pi.SYSTEM_NAME,
        System.getProperty("os.name") );
}
```

As illustrated, the Dano's command code is very clean compared to Jupiter. There are many advantages of this design. For example, if there is a new `SYST2` command based on `SYST`, the new command can inherit from it:

```
Class SYST2 extends SYST{
    Public void runCommand(){
        Super();
        //New code here...
    }
}
```

Another advantage of this design is the commands are separated into different classes. It makes the parts easier to be reused by other applications. For example if a 'stripped-down' version of a FTP server is needed, it only needs to delete the un-needed classes. Comparing this to Jupiter, implementing a stripped-down version of FTP server in Jupiter is not trivial because all of commands are 'tangled' inside the huge `ProtocolInterpreter` class.

Apart from commands, Dano also has many internal polymorphic Factories such as `LoggerFactory` and `FTPFileFactory`. The factories can return different implementations of the same thing based on a standard interface. For example:

```
FTPFile ftpFile = (new
    FTPFileFactory).getFTPFile();
```

It returns a `FTPFileImpl` that conforms to the `FTPFile` interface. That makes the program highly extendable because if a new kind of `FTPFile` type is needed, it only needs to provide a new `FTPFile` implementation and leave the old implementation unchanged. Also this architecture encourages inheritance because a `NewFTPFile` can inherit on the `OldFTPFile`. This encourages potential future reuse and makes the parts of the FTP server more reusable.

6 Evaluation

The polymorphic behaviour index appears to represent what is intended to measure. When inspecting the Lucene internal code, we see Lucene has many types, particularly searching terms and queries, that can be reused in other contexts. This is reflected by a high internal polymorphic behaviour index.

Egothor both extends the API, and internally extends the Query interface providing both external and internal polymorphic behaviour, again reflected by a high polymorphic behaviour index.

For the FTP-servers, our conclusion is that Dano's design features more classes that can be easily reused, both internally and externally.

However there is a problem. Consider the case where a declaration is made such as:

```
Parent obj = new Child();
obj.m();
```

but where the child does not have a method `m()` implementation:

```
class Child1 extends Parent{ ... }
```

The dispatch on `obj.m()` appears to be polymorphic, but is actually non-polymorphic because the Declared Interface is `Parent` but the dispatched method is `Parent.m()`. This case does not occur much in the case studies. However this is a potential problem.

Another issue is the workload used when computing the metric. Different workloads may yield different results. So far we have dealt with this issue by using the metric to just compare between applications in the same domain. Since the same workload is being provided to each application, the comparison does provide useful information. There is still the question, however, as to whether some workloads may give one application an advantage over another. For this we must be careful in choosing representative behaviour in the workloads.

A related issue is the fact that our workloads do not provide complete coverage of the applications' code. This is due to the fact that the applications considered do not have identical functionality, and so a workload that causes 100% coverage of one application will not give complete coverage of the other. As observed above, the code not covered may give that application a disadvantage with respect to our metric.

The polymorphic behaviour metric treats all polymorphic dispatches equally, which means that information about individual dispatches is lost. For example, it may be that a one call site, all dispatches are polymorphic, but at another call site, only some are. It would be interesting to consider a more fine-grained view of polymorphism to examine such behaviour.

7 Conclusions

There are many factors that affect whether or not the benefits claimed of reuse are achieved. In order to gather empirical evidence to support such claims we need to determine how to measure many attributes of software. In this paper we have considered one attribute, namely polymorphism.

We have presented a metric for polymorphic behaviour. We have developed a tool, E-MTRACE, that can measure Java applications based on this metric. We have carried out two case studies that give us confidence that this metric has value, although much more work is needed to validate it.

The polymorphic behaviour metric can provide the basis for other metrics that may be of interest. For example, it can be used to identify the interface that has the most polymorphic dispatches. Such interfaces may form the basis for new APIs or frameworks. Similarly, how many different dispatches a given interface has, or whether the interface has both polymorphic and non-polymorphic

dispatches, may be useful for API development. Finally, there are different ways in which inheritance is used to provide polymorphism. The metric may be extended to distinguish between dispatches due to an internal class inheriting from an API class, versus inheriting from another internal class.

Acknowledgements

We would like to thank the anonymous referees for their comments.

References

- Balda, D. M. and Gustafson, D. A. (1990). "Cost Estimation Models for the Reuse and Prototype Software development lifecycles." ACM SIGSOFT Software Engineering Notes **vol. 15**(3): 42-50.
- Banker, R., Kauffman, J., Wright, C. and Zweig, D. (1994). "Automating output size and reuse metrics in a repository based computer aided software engineering environment." IEEE Transactions on Software Engineering **vol. SE-20**(3): 169-187.
- Barnes, B. H. and Bollinger, T. B. (1991). "Making Reuse Cost Effective." IEEE Software **vol. 8**(1): 13-24.
- Bialecki, A., Cutting, D., Ganyo, S., Goller, C., Gospodnetic, O., Harwood, M., Hatcher, E. and Naber, D. (2005). Lucene project. Apache Project, retrieved from <http://lucene.apache.org/>.
- Choi, K. H. T. (2006). Dynamic Reuse Metrics. Masters Thesis, Electrical and Computer Engineering. Auckland, New Zealand, University of Auckland.
- Clements, P. and Northrop, L. M. (2001). Software Product Lines: Practices and Patterns, Addison Wesley.
- Devanbu, P., Karstu, S., Melo, W. and Thomas, W. (1996). "Analytical and Empirical Evaluation of Software Reuse Metrics." Proceedings of the 18th International Conference on Software Engineering, May.
- DISA/JIEO/CIM (1993). Software Reuse Metrics Plan. Defense Information System Agency, Joint Interoperability Engineering Organization, Center for Information Management, Version 4.1.4.
- Filion, P. (2002). Jupiter-FTP Server. Source forge project, retrieved at <http://jupiter-ftp.sourceforge.net>.
- Frakes, W. and Carol, T. (1994). "Reuse level metrics." Third International Conference on Software Reuse (ICSR '93), Rio de Janeiro, Brazil: 139-148.
- Gaffney, J. E., Jr. and Durek, T. A. (1989). "Software Reuse – Key to enhanced productivity: Some Quantitative models." Information and Software Technology **vol. 31**(5): 258-267.
- Galambos, L. (2006). Egothor Project. Open Source Project, retrieved from <http://www.egothor.org/>.
- Henderson-Sellers, B. (1993). "The Economics of reusing library Classes." Journal of Object-Oriented Programming **vol.6**(4): 43-50.
- Kain, J. B. (1994). "Measuring the ROI of reuse." Object Magazine **vol.4**(3): 48-54.
- Krueger, C. W. (1992). "Software reuse." ACM Computing Surveys **24**(2).
- Malan, R. and Wentzel, K. (1993). Economics of Software Reuse Revisited. Hewlett-Packard Technical Report HPL 93-31.
- McClure, C. (1997). Software Reuse Techniques, Prentice Hall.
- Meyer, B. (1997). Object-oriented software construction (2nd ed.), Prentice-Hall, Inc., Upper Saddle River, NJ, 1997.
- Mili, H., Mili, F. and Mili, A. (1995). "Reusing Software: Issues and Research Directions." IEEE Transactions on Software Engineering **21**(6): 528-561.
- Paregos, M. (2002). DanoFTP Server. Sourceforge Project, retrieved at <http://sourceforge.net/projects/danoftp/>.
- Poulin, J. S. (1997). Measuring Software Reuse, Addison Wesley Longman, Inc, Massachusetts.
- Poulin, J. S. and Caruso, J. M. (1993). "Determining the Value of a Corporate Reuse Program." Proceedings of the IEEE Computer Society International Software Metrics Symposium, Baltimore, MD, 21-22 16-27.
- Tracz, W., Coglianese, L. and Young, P. (1993). "A domain-specific software architecture engineering process outline." SIGSOFT Software Engineering Notes **18**(2): 40-49.

Efficient Cycle-Accurate Simulation of the UltraSPARC III CPU

Peter Strazdins, Bill Clarke and Andrew Over

*Australian National University
sim-devel@ccnuma.anu.edu.au*

Abstract

This paper presents a novel technique for cycle-accurate simulation of the Central Processing Unit (CPU) of a modern superscalar processor, the UltraSPARC III Cu processor. The technique is based on adding a module to an existing fetch-decode-execute style of CPU simulator, rather than the traditional method of fully modelling the CPU microarchitecture. It is also suitable for accurate SMP modelling. The main functions of the module are the simulation of instruction grouping, register interlocks and the store buffer. Its simple table-driven implementation permits easy modification for exploring microarchitectural variations. The technique results in a 40% loss of simulation speed, instead of a 10 times or greater performance loss by fully implementing the detailed micro-architecture. The technique is validated against an actual UltraSPARC III Cu processor, and achieves high levels of accuracy over a range of scientific benchmarks.

1 Introduction

Architectural performance analysis is an increasingly important technique in modern computer systems design (Bose & Conbte 1998). Its main component is called *simulation*, where a model of the system is made; usually this model can reproduce the functional and, in the case of what is termed *execution-driven simulation*, the intended timing behaviour of the system. For *symmetric multiprocessors* (SMPs), the timing accuracy of the simulation is particularly important because the relative timing of events on the different processors affects program behavior as well as performance.

Thus, in order to to perform detailed simulation of a single CPU, or of an SMP system, accurate modelling of the timing characteristics of the CPU is required. This means that the simulated CPU's notion of time, as represented by its clock, must accurately reflect that of the actual processor being simulated.

An example of a project requiring detailed SMP system simulation is in the CC-NUMA Project (Australian National University n.d.). Here accurate performance evaluation of threaded computational chemistry applications is required. As such applications are memory-intensive, detailed memory system simulation (including NUMA effects), down to the explicit modelling of pipelined shared memory transactions, is performed. If the agent injecting

events into the memory system (in this case, the simulated CPU) is not accurate, the timing accuracy of the simulated memory system would be wasted.

Traditionally, cycle-accurate simulation has involved full modelling of the microarchitecture. In a modern, post-RISC CPU, this involves explicit modelling of all stages of the main execution pipeline, that of any sub-pipelines, instruction grouping and (possibly) reordering, and branch prediction logic, together with the resolution of any dependencies between instructions. While this offers potentially the most accurate model, it is substantially more complex to implement and results in a $10 \times$ or more loss of simulation speed (Pai, Ranganathan & Adve 1997, Rosenblum, Bugnion, Devine & Herrod 1997), as compared with the fetch-decode-execute style of CPU simulator.

However, it is possible to accurately predict performance, in terms of the number of clock cycles required for the execution of a sequence of machine instructions, using techniques which only model as much as the CPU's microarchitecture as is necessary for this purpose. In particular, the resource allocation of the functional units, the register interlocks, and the store buffer must be modelled.

For this purpose, 100% accuracy is not necessary; approximations and simplifications can be tolerated, for the sake of performance, provided they have only a small impact on accuracy for situations of interest.

In this paper, we will show how accurate CPU simulation can be implemented as a module which can be added to an existing fetch-decode-execute simulator, called Sparc-Sulima (Clarke, Czezowski & Strazdins 2002, Over, Strazdins & Clarke 2005), of the UltraSPARC III Cu processor (Sun 2002). The implementation is table-driven, and hence can be easily changed to explore the effects of varying instruction execution characteristics.

Related work is discussed in Section 2. Section 3 gives background on relevant aspects of the UltraSPARC III Cu execution pipeline for this work. Section 4 describes the design of the cycle counting module into the framework of Sparc-Sulima, with the validation and performance of the module being given in Section 5. Possible extensions of the module are discussed in Section 6, with conclusions being given in Section 7.

2 Related Work

Related work includes a number of cycle-accurate CPU simulators which simulate the full CPU pipeline, of which the MIPS-based SimOS/MXS (Rosenblum et al. 1997), SPARC-based RSIM (Pai et al. 1997) and SimpleScalar (Burger & Austin 1997) are well known examples. The cost of fully pipelined cycle-accurate simulation in these cases are slowdowns of the order of a thousand in the case of SimpleScalar, and several thousand in the case of SimOS/MXS, as compared

with several hundred for fetch-decode-execution simulators such as SimOS/Mipsy (Rosenblum et al. 1997).

An interesting approach to speed up cycle accurate CPU simulation is the technique of memoization in conjunction with direct execution (Schnarr & Larus 1998). The resulting simulator, FastSim, is reported to be approximately $10 \times$ faster than SimpleScalar (Schnarr & Larus 1998). While this is an impressive result, there are several drawbacks to this approach. Firstly, it is a very complex technique that must be added to an already complex (fully pipelined) simulator¹. Secondly, for memoization to be fully effective, a host processor with a very large memory is required. Thirdly, and most importantly for our purposes, these techniques would lose accuracy if applied in an SMP context. This is because both direct execution and memoization result in the effective skipping of the simulation over a number of cycles. While this permits large speedups to be achieved, it means that accurate interleavings of memory events is not possible.

While such simulators are termed *cycle accurate*, most studies on simulator accuracy compare simulators, or equivalent techniques, with each other, and there are very few studies that actually validate these simulators against real architectures. Indeed, in the case of RSIM and FastSim, the simulator executes a SPARC-like instruction set with a MIPS-based micro-architecture and thus cannot even be calibrated against any existing system. A study on SimOS/MXS configured for the FLASH system against the actual FLASH system has revealed that a cycle accurate simulator may not be accurate at all unless it is validated and debugged against an actual existing system (Gibson, Kunz, Ofelt, Horowitz, Hennessy & Heinrich 2000). The Talisman project is another notable study on simulator validation (Bedichek 1995).

A somewhat similar approach to ours has been made with the Sam CMT Simulator kit (Nussbaum, Fedorova & Small 2004), a series of modules which can be plugged into the SimICS simulator (Virtutech n.d.) in order to simulate UltraSPARC-based chip-multithreaded processors such as the Niagara. While simulation of chip-multithreading seems to be of principal interest, it claims to permit accurate CPU simulation by the means of an *instruction timer* module, which can vary the latency of a particular instruction type. As its purpose is to simulate the Niagara (an 8-core CMP UltraSPARC processor), the module does not model instruction grouping and register dependency effects.

The most similar work to ours we know of is (Loh 2001), where a CPU timing model is based on a general timestamping method for CPU resources (registers and pipelines) was incorporated into a SimpleScalar simulator. The model assumed in-order execution; compared with a an out-of-order model using full pipeline simulation, it was shown to be over two times faster and had an average accuracy (relative to the out-of-order model) of 5% on the SPECint95 benchmarks. However, the SimpleScalar simulator did not model any real architecture. Our work differs in its flexible table-driven design, that it models most of a real CPU including a complex set of grouping rules of and that it has been validated against hardware.

3 UltraSPARC III Cu Instruction Execution Characteristics

The UltraSPARC III Cu is a 4-way superscalar processor, in which instructions are dispatched, but do

not necessarily complete, in program order. Its functional units include 2 ALU pipelines (for the execution of simple integer instructions), a floating-point/graphics multiply (FGM) and add pipeline (FGA), a branch prediction pipeline (BP) and a memory/special instruction (MS) pipeline. Each instruction within a group consumes at least one of the pipelines, and no two instructions can share a pipeline.

Details of this architecture, including a detailed description of features important to this work, such as prefetching, register interlocks and store buffer design, may be found from its manual (Sun 2002).

The MS pipeline is used for complex instructions such as integer multiply and instructions with variable latencies such a load instructions; in this case, the instruction is *recirculated* into the main execution pipeline until the operation completes. These instructions thus have a *blocking latency* $L_b \geq 0$ which blocks the execution of all future instructions for L_b cycles. With the exception of load and store operations, most instructions using the MS pipeline cause the breaking of an instruction group (before or after the instruction), or must execute in a group of their own.

ALU instructions have a latency of one cycle; FGM and and FGM instructions generally have latencies of 3 or 4 cycles. These *locking latencies* only delay the dispatch of future instructions using the output register operand of the current instruction. MS instructions can have a locking latency as well a blocking latency.

The UltraSPARC III Cu has 32 general purpose registers (available in the current register window). It also has 32 double precision floating point registers; the first 16 of these are also used to implement 32 single precision floating point registers.

Load instructions to floating point registers can be serviced from a special prefetch cache (P-cache) instead of the normal top-level data cache. As the P-cache is virtually indexed and tagged, a hit requires no address translation; thus a load floating point instruction can be serviced by one of the ALU pipelines, rather than the MS pipeline. This permits two floating point loads to execute within the same group. The CPU may steer such an instruction if a hit to the P-cache is predicted, i.e. the load hit the P-cache on its previous execution. However, details such as whether the CPU will *always* steer such an instruction, and what occurs on a P-cache miss, are not clear from the documentation (Sun 2002).

4 Design

This section describes the design of an efficient cycle counter module for the UltraSPARC III Cu. As noted in (Strazdins 2005), a few UltraSPARC III features such as block load and store operations are not currently modelled by the cycle counter.

4.1 Incorporation in the Fetch-Decode-Execute Loop

Sparc-Sulima simulates the execution of a program using a fetch-decode-execute ‘run loop’, as indicated in Figure 1. An instruction evaluation function table, indexed by the opcode, is used to perform the execute stage. To improve the speed of the call `DecodeInstr(CI)`, a cache of recently decoded instruction structures is maintained (Clarke et al. 2002). The variable `itracer` is a pointer to an instruction tracing module; when this is non-null, this module can be used to print the current instruction.

¹As a result of this, the authors have begun work on providing automatic support for this approach (Schnarr, Hill & Larus 2001).

```

// pc = value of the Program Counter
CI = FetchInstr(pc);
di = DecodeInstr(CI);
// di is a decoded instruction structure
if (cycleCount)
    // a cycle counter module is installed
    clock += cycleCount->InstrCountCycles(
        di.opcode, &di.operands,
        clock);
else
    clock += 1;
if (itracer)
    // instruction tracing module installed
    itracer->TraceInstr(clock, pc, CI, di);
clock += ExecuteInstr(di.opcode,
    &di.operands);

```

Figure 1: Simplified body of Sparc-Sulima run loop, indicating the incorporation of a cycle counting module

The CPU's clock is simulated by the variable `clock`; this can get updated from a memory system stall upon instruction fetch (not shown in Figure 1) or execution. The clock is also nominally updated by $L = 1$ cycles upon executing each instruction. However, if a cycle counter module is installed (the pointer `cycleCount` is non-null), this latency L can be varied: if the current instruction is the first in the group, $L = L_b + 1$, where L_b was the blocking latency of the previous group. If it is not the first, then $L = 0$ if no dependencies arise through input register availability; otherwise $L > 0$ will be the minimum time for that dependency to be resolved.

Thus, the value of `clock` when `TraceInstr()` (and the instruction evaluation function) is called represents the time when the instruction is actually dispatched for execution. If the instruction is a load or store, this corresponds to the time the memory system first 'sees' the instruction. In this case, the subsequent call to `ExecuteInstr()` may add an extra latency to the instruction cycle.

Figure 2 shows an example instruction trace, where the cycle counter has been installed to update the processor clock. Empty lines separate instruction groups. As the code is highly optimized, a load, floating point multiply and add operation occur on almost every cycle; an exception is at `pc=0x192c8`, where a dependency on register `%f4` from the instruction at `pc=0x1929c` causes a 1 cycle delay. Note that this delays the instruction with the dependency, and all later instructions in the group (Sun 2002, section 4.4.1).

One approximation arises so far in the design: with respect to the memory system, the differences in the value of `clock` between `FetchInstr(pc)` and the instruction evaluation function would be smaller than on a real machine. However, for memory systems having top-level instruction and data caches with independent pathways, this can only make a difference when misses occur to both. In section 6.2, we will describe how this can be overcome, at the cost of introducing some complexity.

An alternative design not requiring a cycle counting module would be for each of the instruction evaluation functions to calculate L and incorporate this in their return values. While this has potential efficiencies in that much of the required information would be more readily available from these functions, this presents software engineering difficulties as many functions would have to be modified (over 300, in the

clock:	pc:	opcode:	operands
297621:	0x19298:	ldd	[%g1],%f2
297621:	0x1929c:	fmuld	%f4,%f18,%f6
297621:	0x192a0:	fadd	%f14,%f8,%f14
297621:	0x192a4:	add	%g1,%o4,%g1
297622:	0x192a8:	ldd	[%g2+%i1],%f18
297622:	0x192ac:	fmuld	%f4,%f20,%f8
297622:	0x192b0:	fadd	%f16,%f10,%f16
297623:	0x192b4:	ldd	[%g2+%i3],%f20
297623:	0x192b8:	fmuld	%f4,%f24,%f10
297623:	0x192bc:	fadd	%f2,%f12,%f2
297624:	0x192c0:	ldd	[%g2+%i0],%f24
297624:	0x192c4:	fmuld	%f4,%f26,%f12
297625:	0x192c8:	fadd	%f4,%f6,%f4
297626:	0x192cc:	ldd	[%i2],%f26
...			

Figure 2: Instruction trace from an optimized matrix-vector multiply program, showing the effect of the cycle counter

case of Sparc-Sulima), and common data structures for the purpose of cycle counting would still have to be maintained by each one. Furthermore, modifying instruction execution characteristics (e.g. for future versions of UltraSPARC) would be difficult.

All latencies associated with the execution of the current instruction are accumulated into the return value of `InstrCountCycles()`. The following sections describe how these latencies may be efficiently computed.

4.2 Table-Driven Design

The following data associated with each instruction needs to be recorded for the purpose of cycle counting: the functional units (pipelines) used by the instruction, whether the instruction causes a group break (before and/or after its execution), its blocking latency (L_b), the latency for its destination register, and the types of the register operands (two source and one destination²).

It is natural to encode this information in tabular form. Noting the large number of instructions (opcodes) for the case of Sparc-Sulima, and observing that many of them share the same characteristics with respect to the above information, our design uses two tables: one mapping opcodes to opcode categories, and another table mapping opcode categories to the above information. While this creates an extra lookup step per instruction, this is a relatively small extra overhead compared with the total required work, and, as there are some 40 opcode categories, this reduces the overall space requirement and so the extra overhead may be offset by reduced cache misses.

However, the real advantage of this design is that it enables easy modification of instruction characteristics for the purpose of microarchitecture exploration, and to an extent makes the cycle counter executable code more microarchitecture-independent.

4.3 Instruction Grouping

The modelling of the instruction grouping by `InstrCountCycles()` amounts to deciding whether the current instruction can be included in the current group.

²A special 'no register' type can be used to represent a missing operand.

This involves firstly determining whether it can be included according to whether all required functional units (pipeline) resources are available. A bitmask of all functional units used so far by the current group is maintained by the cycle counter; a simple 'logical or' of this mask with the current instruction's functional units mask entry in the table determines this. The only complication is that there are two ALU pipelines. This is resolved by having a single ALU bit field in the current group's bitmask; this field is only set by the second instruction requiring an ALU pipeline, through the use of a flag that is set by the first.

Secondly, the group breaking characteristics of the current instruction are examined to determine if a break of a group should occur before and/or after the current instruction.

After a group breakage occurs, the maximum of all blocking latencies for the group (plus 1) is returned by the function.

For each instruction, typically only 11 load/store operations, 4 comparisons and 10 arithmetic/logic operations are required.

The bitmask implementation implicitly assumes each instruction locks its required functional units for at most one cycle, unless it has a blocking latency of $L_b > 0$, in which case it locks *all* functional units for another L_b cycles. This is valid for all UltraSPARC III Cu instructions except for the floating point divide and square root, which only lock the FGM pipeline for L_b cycles. This limitation could be overcome at the loss of some efficiency by using timestamps for each functional unit, in a similar way as it is done for register resources, as will be described in Section 4.5.

4.4 Prefetching and Floating Point Load Steering

As mentioned in Section 3, floating point loads can be steered to an ALU pipeline if a hit to the P-cache is predicted. This can affect the performance of floating point-intensive codes significantly. As we have recently implemented a detailed memory model for the UltraSPARC III Cu including the P-cache (Over et al. 2005), we are able to implement this feature in the cycle counter.

In our implementation, steering only happens if the MS pipeline is already used, and there is an ALU pipeline free, and a load has not been steered in this group. If a floating-point load gets steered, a flag is marked by the cycle counter. Upon evaluation of the load instruction, this flag is checked. If the flag is set, only the P-cache is checked. If the P-cache misses, a special exception is triggered which recirculates the instruction (including going through the cycle counter again). Since the current instruction group already uses the MS pipeline and an ALU pipeline with a steered load, the recirculated load will go in a new group.

4.5 Register Interlocks

Register interlocks is a term denoting the mechanism determining any extra latency cycles due to inter-instruction dependencies from register resource requirements.

Our design needs to be efficient on highly optimized floating point codes (see Figure 2). In such cases, with up to three instructions per group that may be updating a floating point register with a maximum latency of $L_{\max} = 4$ cycles, there could be as many as 12 floating point registers at any time which are *interlocked*, i.e. unavailable due to pending operations.

Under these circumstances, maintaining a list of registers currently interlocked (together with a list of timestamps when the interlocks expire) would be expensive, requiring traversal and compaction this list.

A preliminary solution, described in detail in (Strazdins 2005) was to use bitmasks to represent the interlocks; since registers can be locked for up to $L_{\max} = 4$ cycles in the future, a circular array of length L_{\max} of two 64-bit bitmasks suffices. This requires some care, with the structure needing to be updated with the introduction of any latencies.

A simpler and more general solution (suitable for large L_{\max}) is have individual timestamps for each register; the timestamp signifies the time when the register will first become available. These timestamps are organized as an array indexed by the register file type (GPR, single and double floating point), and the index of the register within that file³.

The cycle counter uses these timestamps as follows. Let t denote the current time (inside `InstrCountCycles()`, this will be the value of the `clock` parameter, with any latencies from the previous group added). If a source register of the current instruction has timestamp $t' > t$, t is updated to t' (thus introducing a latency of $t' - t$). The destination register's time stamp is then set to $t' + L_b + L$, where L_b is the current instruction's blocking latency and L is its locking latency.

The worst-case overhead (realized in the case of a floating point arithmetic instruction) involves approximately 10 load, 10 comparison and 2 store operations.

4.6 Store Buffer

While its documentation suggests that the UltraSPARC III Cu processor can sustain one store operation per cycle (Sun 2002), at least under ideal conditions (e.g. for computations with working sets fitting in the top-level caches and having unit stride memory access patterns), a series of store-intensive benchmarks with double precision data indicated that the maximum sustained rate was one store every three cycles. Thus, to gain accuracy for these computations, a store buffer having a draining rate of one store per $\Delta t_S = 3$ cycles was required.

Such a buffer can be efficiently modelled as follows. The values t_S , the time the buffer last emitted a store operation, and l_S , the number of pending stores in the buffer, are maintained. The maximum depth of the store buffer is $l_S^{\max} = 8$ (Sun 2002). Upon a store operation, the following is performed. If $t_S < t$, the store buffer can be brought up-to-date by draining $n_S = \lfloor \frac{t - t_S}{\Delta t_S} \rfloor$ stores from the buffer (l_S is decremented by n_S) and incrementing t_S by $n_S \Delta t_S$.

If the store buffer is full ($l_S = l_S^{\max}$), t_S is bought up to the next emission time ($t_S \leftarrow t_S + \Delta t_S$), and now a latency of $\delta = t_S - t$ is introduced (δ is added to the return value of `InstrCountCycles()`). Otherwise, the store is merely added to the buffer (by incrementing l_S), and no other action need be taken.

4.7 Branch Prediction

On the UltraSPARC III Cu, a branch misprediction occurs when a (conditional) branch instruction computes a different target address to what is predicted

³Treating the single and double precision registers as being in separate files is a simplification introduced here, deemed acceptable since codes using single and double data within the same registers over very short time intervals are rare. Exact modelling could be achieved by having individual timestamps for the upper and lower halves of each double precision register, at the expense of extra implementation overhead.

(the target address of its previous execution, or, if fresh to the I-cache, the initial prediction bit of the instruction). Misprediction causes a group breakage (after the delay slot instruction) and adds an extra blocking latency (8 cycles) to the group containing the branch.

The decoded instruction operands structure is used to record a branch instruction's prediction bit. When a branch instruction is passed to `InstrCountCycles()`, its operands structure's address is recorded. Whether the branch is taken or not is then recorded in the main run loop, and this information is passed to `InstrCountCycles()` with the next instruction (the branch's 'delay slot'). At this point, the cycle counter deals with a misprediction (if one occurred), and stores whether the branch was taken in the branch's operands structure for future use.

4.8 Counting CPU-Specific Events

The UltraSPARC III Cu has hardware support for counting events for the purposes of performance analysis. For the purposes of validating the simulator, it is important to implement these as well (Over et al. 2005). The cycle counter module has sufficient information to monitor CPU-related events such as counts of instruction executed, elapsed cycles, and FGM and FGA instructions executed (using the bit-masks for functional units usage in the instruction category table). Stall cycles from GPR dependencies, floating point register dependencies and a full store buffer can also be easily counted by the module.

This functionality proved important in the validation of the cycle counter.

5 Evaluation

In this section, we will evaluate the accuracy of the cycle counter module and its effect on simulation performance, using optimized numerical programs as benchmarks.

5.1 Validation

The first stage of validation is to manually examine instruction traces such as that in Figure 2 and see if the timestamps are consistent with the rules provided in the UltraSPARC III Cu manual (Sun 2002). This tests whether the module is accurate with respect to the CPU, as defined by its documentation.

However, the timing of real implementation of the CPU may be different to what is specified in its documentation (usually slower); so the next stage involves comparison of benchmark performance when run on the simulator and on the real machine. When discrepancies were detected, comparisons of performance counters for the simulator and the real hardware and analysis the disassembled code of the benchmark were used to gain some insights, and often microbenchmarks were constructed to verify the suspected cause.

Two intricacies in the UltraSPARC grouping rules took a long time to discover. The 'same group bypass' rule states that "no instruction can bypass the result to another instruction in the same group" (Sun 2002, Section 4.4.4). Since all result-producing instructions have a latency of at least 1 cycle, this was initially interpreted that the instructions were to be grouped together, but the second instruction was stalled till the result was ready. This can be different to the situation where a group break is forced before the second instruction, which from microbenchmarks we believe to be the correct interpretation.

The second relates to grouping behaviour when a register dependency stall is encountered. The relevant documentation states that "the offending instruction and all younger instructions [already in the group] are recirculated (Sun 2002, Section 4.4.3). It does not specify if regrouping (with new instructions) is possible. It was initially assumed that it was not; however, this caused the cycle counter to be pessimistic by about 15% on optimized floating-point intensive codes such as matrix-vector multiply. Running the code with performance counters on the real hardware indicated that much fewer than expected stall cycles from register dependencies was occurring. Inspection of the assembly code indicated that if the grouping was such that the load instructions were scheduled at the bottom rather than at the top of the group (see Figure 2), this would separate the load with the dependent multiply by another cycle, eliminating the stall. Implementing regrouping in the cycle counter gave such a pattern, as the first multiply causing a stall (in the 24-way unrolled loop) caused the load instructions in subsequent groups in the loop to be scheduled at the bottom. With this, the cycle counter agreed closely with both the performance counters and the execution time of the real hardware.

We first present a set of mini-benchmark results from simple matrix-vector operations (the algorithm used for $y \leftarrow A^{-1}x$ was an iterative solver based on matrix-vector multiply).

The benchmark codes were compiled without prefetch instructions, but otherwise high levels of compiler optimizations were performed. The simulator was compiled in 64-bit mode under GCC 3.4.4 with optimization flags `-x02`. To minimize the effects of the memory system on simulated time, the simulator was configured with the store buffer, prefetch cache and write cache disabled.

Table 1 gives results for vector operations; the computations were repeated a sufficient number of times (normally 100 – 10000×) for accurate timings with a high resolution timer, after both the instruction and data caches were warmed. As the execution time is thus dominated by a relatively small loop body, these can be classified as micro-benchmarks. The vector operation results are expressed in terms of MOPs, which is millions of vector elements processed per second. Note that the working set fit within the top-level caches in all cases. The results show that for store-intensive computations, modelling the store buffer substantially improved accuracy. With the store buffer, the results are always optimistic and within 33% accuracy (in most cases, well within).

A comparison of performance counters for the simulator and real hardware indicate that store-buffer (and write-cache) effects accounted for the bulk of these discrepancies. The table also indicates the effect of using a cycle-accurate memory system model (Over et al. 2005). This model detailed store-buffer and write-cache modelling and deals with issues such as read-after-write hazards, coalescing, and write-cache bandwidth. The memory model has been extensively validated using memory-intensive microbenchmarks (Over et al. 2005); despite this, it tends to be pessimistic on these particular benchmarks. It has proved very difficult to develop a store-buffer and write-cache model that is accurate in all situations.

The cycle counter, together with cycle-accurate memory model, has also been validated over the NAS Parallel Benchmarks (NASA Advanced Supercomputing n.d.) (OMP version, using one thread). Table 2 gives comparisons for the main computation phase of the benchmarks. The average absolute error is 4.4%, the average error is -2.1% (the simulator is generally optimistic), the maximum error

computation	simulated				actual
	no CC	CC-SB	CC+SB	CC+SB'/WC	
$y \leftarrow x$ ($8 \times \text{ldd}; 8 \times \text{std}$)	309	445	298	224	297
$y \leftarrow x$ ($1 \times \text{ldd}; 1 \times \text{std}$)	150	299	299	224	224
$y \leftarrow 2x$	123	216	148	111	136
$y \leftarrow y + x$	170	288	288	222	254
$y \leftrightarrow x$	169	221	149	108	118
$a \leftarrow \sum_{i=0}^{n-1} x_i $	179	179	179	179	179
$x \leftarrow 2x$	210	446	298	224	256
$x \leftarrow 0$	497	866	296	223	292
$y \leftarrow Ax$	476	1376	1376	1376	1309
$y \leftarrow A^{-1}x$	442	1141	1158	1141	1062

Table 1: Speed in MOPs for vector-vector computations (length 4000) and in MFLOPs for vector-matrix computations (A is 64×64 and row-major) on an 900 MHz UltraSPARC III Cu (CC = cycle counter, SB = store buffer, SB'/WC = detailed store buffer and write-cache)

being 12%. These results are generally better than in Table 1, due to the overall intensity of store operations being less. The results of this section together thus indicate a high degree of accuracy of the cycle counter over a range of scientific applications, with the store-intensive mini-benchmarks exposing a particularly difficult situation in the validation of an UltraSPARC CPU and memory system.

5.2 Performance

Table 3 indicates the overheads of the cycle counter module. These measurements were based on execution time (process user time) of the simulator running the whole test program, with the indicated computation being repeated 1000 times in order to ensure other parts of the program (e.g. data initialization and result checking) made a negligible contribution. The column “no CC: CC-RIL-LS” indicates the slowdown of the simulator with a cycle counter module with register interlock checking and floating point load steering suppressed over that of the simulator with no module, whereas the column “no CC: CC-LS” indicates that with only load steering suppressed. The column “no CC: CC” indicates the slowdown of the simulator with a complete cycle counter module over that of the simulator with no module.

The column “host: no CC” indicates the slowdown of simulator program (without a cycle counter module) over the same test program being run natively on an UltraSPARC III Cu. This gives an indication of simulator efficiency. However, it is profoundly influenced by the speed (degree of instruction-level parallelism) of the computation when run natively. For example, the simulator’s slowdown for a LINPACK Benchmark program with $n = 500$, which ran at 640 MFLOPs, was 544; thus the matrix-vector multiply, running at twice as many MFLOPs, has approximately twice as large a slowdown.

These results indicate that computing instruction groupings adds 10–20% overhead, and that computing register interlocking adds a further 10–20% overhead. Floating point load steering adds a variable amount of overhead, with the total cycle counter overhead being about 40%. For the LINPACK computation mentioned above, the overheads were similar to that of $y \leftarrow Ax$.

FastSim achieves slowdowns of between 150–300 on the SPEC95 Benchmarks (Schnarr & Larus 1998). In terms of CPI and the frequency of memory system events (which are expensive to simulate), the $a \leftarrow \sum_{i=0}^{n-1} |x_i|$ computation above would be the closest match to these. Taking this into account, FastSim seems to be faster by a factor of at least two. However, the results are not directly comparable since while

both simulators are SPARC-based, SparcSulima simulates a much larger ISA, which adds a significant degree of overhead to the basic simulator (Clarke et al. 2002), and the FastSim slowdowns are based on a much earlier host system (a 167 MHz UltraSPARC I). As simulation is memory-intensive, it is advantageous in terms of slowdowns to use a host with a lower memory latency with respect to CPU speed.

6 Extensions to the Design

This section indicates how the design of Section 4 may be improved and extended to broader contexts.

6.1 Performance Enhancement via Caching

The *caching* of expensive computations, such as the decoding of instructions, has been an important technique in improving the performance of SparcSulima (Clarke et al. 2002). Note that memoization (Schnarr & Larus 1998) is an extension of this technique. In the case of the cycle counter, one way of applying this idea would be to record the result (the accumulated latency) of the previous call to `InstrCountCycles()` with a given instruction in a lookup table. The table would be tagged and indexed by the instruction’s address, with the `pc` now needing to be passed to the function as well.

This result could be re-used on the next execution of this instruction if it were known that the conditions over the last L_{\max} cycles were identical on the previous execution. A sufficient condition for this would be whether the instruction groupings were identical over the last L_{\max} cycles. In turn, this can be determined from whether the values of the `pc` were the same over the last L_{\max} cycles (if an instruction caused a multi-cycle stall, the corresponding `pc` value would be repeatedly recorded; if multiple instructions were issued in the one cycle, the first could be used).

This information would be maintained `InstrCountCycles()` for the current instruction, and also stored in the lookup table. To save space and improve lookup speed, a 64-bit hash value (e.g. for the case $L_{\max} = 4$, the bits 17:2 of the `pc`’s) could be used.

On a hit, the lookup table’s latency would be returned, without having to maintain the cycle counter’s data structures for determining instruction groupings and register interlocks. The overhead should be quite small.

The difficulty arises upon the first miss, as these data structures have to be reconstructed. In principle this would be possible, provided the cycle counter also maintained a list of all parameters (clock values, opcode and decoded operand structure addresses) over

workload	bt.S	ft.S	is.S	lu.S	lu-hp.S	mg.S	sp.S
ratio host:sim.	1.00	1.07	0.88	1.00	0.98	0.90	1.00

Table 2: NAS Benchmark validation

computation	host: no CC	no CC: CC-RIL-LS	no CC: CC-LS	no CC: CC
$y \leftarrow x$	966	1.07	1.21	1.35
$y = y + x$	1435	1.03	1.22	1.42
$a \leftarrow \sum_{i=0}^{n-1} x_i $	590	1.19	1.36	1.39
$y \leftarrow Ax$	1757	1.17	1.27	1.41

Table 3: Slowdowns for selected computations of Table 1 indicating overall speed of simulator and added overheads due to the cycle counter module, for an 900 MHz UltraSPARC III Cu (CC = cycle counter, RIL = register interlocks)

the last L_{\max} cycles, and ‘replayed’ these instructions in order to perform the reconstruction.

However, the fact that the cycle counter must maintain state data persisting over a considerable number of invocations makes it less amenable to the computation caching technique. Furthermore, branch and P-cache load mis-predictions (Sections 4.7 and 4.4) would raise further complications. For these reasons, and the fact that the cycle counter’s overheads has been acceptable (within 50%), this has not yet been implemented.

6.2 Improving Fetch-Execute Timing Accuracy and Extension to Out-of-order Execution

As mentioned in Section 4, a minor source of inaccuracy in the current design is that the timestamps, as would be seen by the memory system, between the fetch and execute stage of a load or store instruction would appear closer than on the real machine, due to the number of pipeline stages (5 in the case of the UltraSPARC III) between these stages.

Furthermore, while UltraSPARC-specific behaviour is mainly confined to the tables, one implicit assumption is that instructions always execute in-order. This prevents the design from easily being adapted to other post-RISC processors, such as the MIPS and Alpha.

These limitations could be overcome by a cycle counter design that ‘buffered’ a number of instructions. Upon each invocation, it would firstly choose which instruction in its buffer is to be executed, then calculate as before its latency contribution, and then emit the instruction (corresponding pc value, opcode and decode operands) for the instruction evaluation function and the rest of the main run-loop.

This of course would add complexity, particularly if the reordering rules are complex, and considerable overhead, particularly if a large instructions buffer is needed. However, with this extension, a modular cycle counter design could still be used to capture CPU timing behaviour without needing to resort to full microarchitecture simulation.

6.3 Integration into a SimICS-like Simulator

SimICS is a complete-machine simulator that has been used in many places to simulate UltraSPARC-based systems (see e.g. (Nussbaum et al. 2004)). The core SimICS simulator provides ‘hooks’ for user-defined modules to be plugged in (Virtutech n.d.) and called upon certain events, such as the execution of each instruction.

The cycle counter design presented so far is independent of the other aspects of Sparc-Sulima, except

in the decoded instruction operand structure. Its efficiency also relies heavily on having the decoding available. Thus, a ‘wrapper’ function would be needed to be called from SimICS instead; this function would take the un-encoded instruction as its parameter, maintain its own *instruction decode cache* (Clarke et al. 2002), and then call `InstrCountCycles()` with the result of the decodings.

7 Conclusions

The cycle counter presented above substantially improved the timing accuracy of CPU aspects of an UltraSPARC simulator with only modest implementation overhead (< 40%) and complexity of implementation, compared with the traditional approach of fully modelling the CPU microarchitecture. It thus represents a ‘sweet-spot’ in the accuracy / performance tradeoff that is central to computer simulation design. It is suitable for coupling with cycle-accurate SMP memory system models, enabling a realistic timestamping of events passed to the simulated memory system. The two main features of its implementation, determination of instruction groups and that of stall cycles due to register dependencies, contributed approximately equally to the overhead.

Its modular design permits it to be activated only in the parts of the simulation where timing is important. Most platform-specific information is recorded in tables, permitting the easy exploration of varying timing-related characteristics of the microarchitecture in order to determine its effect on overall application performance. The algorithms and methods used can also be reasonably easily adapted to other post-RISC architectures, except that modelling out-of-order execution would require considerable extension.

While the approach we have taken is mainly applicable to in-order execution, we believe that the advent of chip-multiprocessing and the increasing concern over power consumption will favour the introduction of in-order cores, particularly for heterogeneous chip-multiprocessing (Kumar, Tullsen, Jouppi & Ranganathan 2005).

Manual inspection of the timings made by the cycle counter demonstrate that it closely implements the main documented timing characteristics of the UltraSPARC III Cu. However, in order to model store-intensive applications (even when the working set can be kept within the top-level cache), undocumented information on the store buffer had to be reverse engineered by experiments. Without validation against a real system, it would have been difficult to detect this large source of inaccuracy.

In the benchmarks of interest, the cycle counter proved highly accurate compared with the real ma-

chine. The main discrepancies were on store-intensive benchmarks where neither the cycle-counter's simple model, nor a highly sophisticated memory system model developed elsewhere, could model the real hardware to full accuracy in all situations.

A few aspects of the CPU are not currently modelled. It is expected that each new feature could add a small amount of overhead. In principle, this would broaden the range of applications over which the simulator is accurate. However, the real limiting factor in improving simulation accuracy is the lack of complete and accurate performance-related information available on the real, as opposed to the documented, version of the CPU.

The source code for the SparcSulima simulator, including the cycle counter, is available from <http://ccnuma.anu.edu.au/sulima/>.

Acknowledgements

The authors thank the Australian Research Council, Sun Microsystems Inc. and Gaussian Inc. This research has been funded under the ARC Linkage Grant LP0347178.

References

- Australian National University (n.d.), 'The CC-NUMA project: Computational Chemistry on Non-Uniform Memory-access Architectures', <http://cs.anu.edu.au/CC-NUMA>.
*<http://cs.anu.edu.au/CC-NUMA>
- Bedichek, R. C. (1995), Talisman: Fast and Accurate Multicomputer Simulation, in 'Proceedings of the 1995 ACM SIGMETRICS Joint International Conference on Measurement and Modelling of Computer Systems', ACM Press, pp. 14–24.
- Bose, P. & Conbte, T. M. (1998), 'Performance Analysis and its Impact on Design', *IEEE Computer* pp. 41–49.
- Burger, D. & Austin, T. (1997), The SimpleScalar Tool Set, Version 2.0, Technical Report TR-1342, University of Wisconsin-Madison Computer Sciences Department.
- Clarke, B., Czezowski, A. & Strazdins, P. (2002), Implementation aspects of a SPARC V9 complete machine simulator, in M. Oudshoorn, ed., 'Computer Science 2002', Vol. 4 of *Conferences in Research and Practice in Information Technology*, Australian Computer Society, Monash University, Melbourne, pp. 23–32.
- Gibson, J., Kunz, R., Ofelt, D., Horowitz, M., Hennessy, J. & Heinrich, M. (2000), FLASH vs. (Simulated) FLASH: Closing the Simulation Loop, in 'Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems', ACM Press, pp. 49–58.
- Kumar, R., Tullsen, D. M., Jouppi, N. P. & Ranganathan, P. (2005), 'Heterogenous Chip Multiprocessors', *IEEE Computer* **38**(11), 32–38.
- Loh, G. (2001), A Time-Stamping Algorithm for Efficient Performance Estimation of Superscalar Processors, in 'Proceedings of the 2001 ACM SIGMETRICS Joint International Conference on Measurement and Modelling of Computer Systems', ACM Press, pp. 72–81.
- NASA Advanced Supercomputing (n.d.), 'NAS Parallel Benchmarks', <http://www.nas.nasa.gov/Software/NPB/>. Version 3.1.
*<http://www.nas.nasa.gov/Software/NPB/>
- Nussbaum, D., Fedorova, A. & Small, C. (2004), An overview of the Sam CMT simulator kit, Technical Report TR-2004-133, Sun Microsystems Research Labs.
*<http://research.sun.com/techrep/2004/abstract-133.html>
- Over, A., Strazdins, P. & Clarke, B. (2005), Cycle-Accurate Memory Modelling: A Case-Study in Validation, in 'Proceedings of the 13th International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems', IEEE, pp. 85–94.
- Pai, V. S., Ranganathan, P. & Adve, S. V. (1997), RSIM: An Execution-Driven Simulator for ILP-Based Shared-Memory Multiprocessors and Uniprocessors, in 'Proceedings of the Third Workshop on Computer Architecture Education'. Also appears in IEEE TCCA Newsletter, October 1997.
- Rosenblum, M., Bugnion, E., Devine, S. & Herrod, S. (1997), 'Using the SimOS Machine Simulator to Study Complex Computer Systems', *ACM TOMACS Special Issue on Computer Simulation*.
- Schnarr, E. C., Hill, M. D. & Larus, J. R. (2001), Facile: a language and compiler for high-performance processor simulators, in 'PLDI '01: Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation', ACM Press, pp. 321–331.
- Schnarr, E. & Larus, J. R. (1998), Fast out-of-order processor simulation using memoization, in 'Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems', ACM Press, New York, New York, pp. 283–294.
- Strazdins, P. (2005), CycleCounter: an Efficient and Accurate UltraSPARC III CPU Simulation Module, Technical Report TR-CS-05-01, Department of Computer Science, Australian National University.
- Sun (2002), *UltraSPARC III Cu User's Manual*.
- Virtutech (n.d.), 'Simics 1.0', <http://www.simics.com/>.
*<http://www.simics.com/>

Author Index

- Ancona, Massimo, 25
 Askitis, Nikolas, 97
- Bell, Genevieve, 3
 Billard, Angela, 181
 Bogdanovych, Anton, 25
 Boyd, Colin, 191
- Carrington, David, 171
 Choi, Kelvin (Hio Tong), 211
 Citro, Sandy, 115
 Clarke, Bill, 221
 Cooper, James, 161
 Corman, Amy, 35
- Dobbie, Gillian, iii
 Drago, Sara, 25
- Estivill-Castro, Vladimir, 141
- Fenwick, Joel, 141
 Fidge, Colin, 171
 Foo, Ernest, 191
- Goldman, Grigori, 131
 González Nieto, Juan Manuel, 191
- Hao, Yanan, 107
 Hartmann, Sven, 69
- Indulska, Jadwiga, 49
- Jeremic, Nikifor, 181
- Kelly, Wayne, 41
 Koehler, Henning, 79
- Link, Sebastian, 69
- Liu, Chuchang, 181
 Long, Benjamin, 171
- Marriott, Kim, 7
 Mason, Richard, 41
 McGovern, Jim, 115
 Melton, Hayden, 87, 201
- Over, Andrew, 221
 Ozols, Maris, 181
- Pirzada, Asad, 49
 Portmann, Marius, 49
- Ritter, Nicola, 161
 Ryan, Caspar, 115
- Sbarski, Peter, 7
 Schachte, Peter, 35
 Sierra, Carles, 25
 Simoff, Simeon, 25
 Sinha, Ranjan, 97
 Song, Jian, 125
 Strazdins, Peter, 221
- Tahaghoghi, Seyed M.M., 151
 Takaoka, Tadao, 15
 Teague, Vanessa, 35
 Tempero, Ewan, 87, 201, 211
 Thom, James A., 151
 Tian, Lin, 15
 Tritilanunt, Suratoose, 191
- Volkmer, Timo, 151
- Zhang, Yanchun, 107, 125
 Zhao, Ying, 59
 Zobel, Justin, 59

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 53 - Conceptual Modelling 2006

Edited by Markus Stumptner, *University of South Australia*, Sven Hartmann, *Massey University, New Zealand* and Yasushi Kiyoki, *Keio University, Japan*. January, 2006. 1-920-68235-X.

Contains the proceedings of the Third Asia-Pacific Conference on Conceptual Modelling (APCCM2006), Hobart, Tasmania, Australia, January 2006.

Volume 54 - ACSW Frontiers 2006

Edited by Rajkumar Buyya, *University of Melbourne*, Tianchi Ma, *University of Melbourne*, Rei Safavi-Naini, *University of Wollongong*, Chris Steketee, *University of South Australia* and Willy Susilo, *University of Wollongong*. January, 2006. 1-920-68236-8.

Contains the proceedings of the Fourth Australasian Symposium on Grid Computing and e-Research (AusGrid 2006) and the Fourth Australasian Information Security Workshop (Network Security) (AISW 2006), Hobart, Tasmania, Australia, January 2006.

Volume 55 - Safety Critical Systems and Software 2005

Edited by Tony Cant, *University of Queensland*. April, 2006. 1-920-68237-6.

Contains the proceedings of the 10th Australian Workshop on Safety Related Programmable Systems, August 2005, Sydney, Australia.

Volume 56 - Vision in Human-Computer Interaction

Edited by Roland Goecke, Antonio Robles-Kelly, and Terry Caelli, *NICTA*. November, 2006. 1-920-68238-4.

Contains the proceedings of the HCSNet Workshop on the Use of Vision in Human-Computer Interaction (VisHCI 2006).

Volume 57 - Multimodal User Interaction 2005

Edited by Fang Chen and Julien Epps *National ICT Australia*. April, 2006. 1-920-68239-2.

Contains the proceedings of the NICTA-HCSNet Multimodal User Interaction Workshop 2005, Sydney, Australia, 13-14 September 2005.

Volume 58 - Advances in Ontologies 2005

Edited by Thomas Meyer, *National ICT Australia, Sydney* and Mehmet Orgun *Macquarie University*. December, 2005. 1-920-68240-6.

Contains the proceedings of the Australasian Ontology Workshop (AOW 2005), Sydney, Australia, 6 December 2005.

Volume 60 - Information Visualisation 2006

Edited by Kazuo Misue, Kozo Sugiyama and Jiro Tanaka. February, 2006. 1-920-68241-4.

Contains the proceedings of the Asia-Pacific Symposium on Information Visualization (APVIS 2006), Tokyo, Japan, February 2006.

Volume 61 - Data Mining and Analytics 2006

Edited by Peter Christen, *Australian National University*, Paul J. Kennedy, *University of Technology, Sydney*, Jinyong Li, *University of Southern Queensland*, Simeon Simoff, *University of Technology, Sydney* and Graham Williams, *Australian Taxation Office*. December, 2006. 1-920-68242-2.

Contains the proceedings of the Australasian Data Mining Conference (AusDM 2006), Sydney, Australia. December 2006.

Volume 62 - Computer Science 2007

Edited by Gillian Dobbie, *University of Auckland, New Zealand*. January, 2007. 1-920-68243-0.

Contains the proceedings of the Thirtieth Australasian Computer Science Conference (ACSC2007), Ballarat, Victoria, Australia, January 2007.

Volume 63 - Database Technologies 2007

Edited by James Bailey, *University of Melbourne* and Alan Fekete, *University of Sydney*. January, 2007. 1-920-68244-9.

Contains the proceedings of the Eighteenth Australasian Database Conference (ADC2007), Ballarat, Victoria, Australia, January 2007.

Volume 64 - User Interfaces 2007

Edited by Wayne Piekarski, *University of South Australia*. January, 2007. 1-920-68245-7.

Contains the proceedings of the Eighth Australasian User Interface Conference (AUIC2007), Ballarat, Victoria, Australia, January 2007.

Volume 65 - Theory of Computing 2007

Edited by Joachim Gudmundsson, *NICTA, Australia* and Barry Jay *UTS, Australia*. January, 2007. 1-920-68246-5.

Contains the proceedings of the Thirteenth Computing: The Australasian Theory Symposium (CATS2007), Ballarat, Victoria, Australia, January 2007.

Volume 66 - Computing Education 2007

Edited by Samuel Mann, *Otago Polytechnic* and Simon Newcastle *University*. January, 2007. 1-920-68247-3.

Contains the proceedings of the Ninth Australasian Computing Education Conference (ACE2007), Ballarat, Victoria, Australia, January 2007.

Volume 67 - Conceptual Modelling 2007

Edited by John F. Roddick, *Flinders University* and Annika Hinze, *University of Waikato, New Zealand*. January, 2007. 1-920-68248-1.

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, *University of Auckland*, and Andrew Wendelborn, *University of Adelaide*. January, 2006. 1-920-68249-X.

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 72 - Advances in Ontologies 2006

Edited by Mehmet Orgun *Macquarie University* and Thomas Meyer, *National ICT Australia, Sydney*. December, 2006. 1-920-68253-8.

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. 1-920-68254-6.

Contains the proceedings of the AI 2006 Workshop on Intelligent Systems for Bioinformatics (WISB-2006), Hobart, Australia, December 2006.