Conferences in Research and Practice in
Information Technology

Volume 74

Computer Science 2008

Australian Computer Science Communications, Volume 30, Number 1

Computer Science 2008

Proceedings of the
Thirty-First Australasian Computer Science Conference
(ACSC 2008), Wollongong, NSW, Australia,
January 2008

Gillian Dobbie and Bernard Mans, Eds.

Volume 74 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.

Computer Science 2008. Proceedings of the Thirty-First Australasian Computer Science Conference (ACSC 2008), Wollongong, NSW, Australia, January 2008

Conferences in Research and Practice in Information Technology, Volume 74.

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

Bernard Mans
Department of Computing
Division of Information and Communication Sciences
Macquarie University
Sydney, NSW 2109
Australia
Email: bmans@ics.mq.edu.au

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

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

Conferences in Research and Practice in Information Technology, Volume 74
ISSN 1445-1336
ISBN 978-1-920682-55-2

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

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

Table of Contents

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 Thirty First ACSC in Wollongong, NSW, Australia. ACSC 2008 is part of the Australasian Computer Science Week which ran from Jan 22nd to 25th, 2008.

The ACSC 2008 call for papers solicited contributions in all areas of computer science research. This years conference received submissions from Australia, New Zealand, China, France, India, Iran, Jamaica, Jordon, Malaysia, Pakistan, South Africa, Turkey, UK, 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 (9 papers)
- Artificial Intelligence (7 papers)
- Communications and Networks (4 papers)
- Computer Architecture (2 paper)
- Computer Vision (4 papers)
- Databases (5 papers)
- Distributed Systems (6 papers)
- E-Commerce (4 papers)
- Formal Methods (6 papers)
- Graphics (6 papers)
- High Performance Computing (7 papers)
- Human-Computer Interaction (8 papers)
- Mobile Computing (6 papers)
- Multimedia (1 paper)
- Object Oriented Systems (3 papers)
- Ontologies (1 paper)
- Operating Systems (5 papers)
- Programming Languages (4 papers)
- Robotics (1 paper)
- Scientific Computing (5 papers)
- Security and Trusted Systems (5 papers)
- Simulation (6 papers)
- Software Engineering (5 papers)
- Speech (1 paper)
- Theory (3 papers)
- Visualization (6 papers)
- Web Services (3 papers)

The programme committee consisted of 28 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 47 papers submitted, 16 were selected for presentation at the conference.

The programme committee invited Professor Joxan Jaffar, to give a keynote on Constraint Logic Programming for Program Analysis. Professor Jaffar has recently completed a stint as Dean of the School of Computing from 2001-2007 at the National University of Singapore. His interests are in programming languages and applications, with emphasis on the logic and constraint programming paradigms. Amongst his main contributions are the principles of constraint logic programming, and the widely-used CLP(R) system. The committee also invited Dr Benjamin Burton and Associate Professor Ewan Tempero to give invited talks. Dr Burtons talk was entitled Informatics Olympiads:Challenges in Programming and Algorithm Design. Associate Professor Temperos talk is entitled On Measuring Java Software.

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 Wollongong.

Gillian Dobbie
University of Auckland

Bernard Mans
Macquarie University

ACSC 2008 Programme Chairs
January 2008

Programme Committee

Chairs

Gillian Dobbie, University of Auckland, New Zealand
Bernard Mans, Macquarie University, Australia

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
Sally Jo Cunningham, University of Waikato, New Zealand
Gillian Dobbie, University of Auckland, New Zealand
Jenny Edwards, University of Technology, Sydney, Australia
Colin Fidge, Queensland University of Technology, Australia
Aditya Ghose, University of Wollongong, Australia
Ken Hawick, Massey University - Albany, New Zealand
Nigel Horspool, University of Victoria, Canada
Michael Houle, National Institute for Informatics, Japan
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
Michael Oudshoorn, Montana State University, US
Masahiro Takatsuka, The University of Sydney, Australia
Bruce H. Thomas, University of South Australia, Australia
Andrew Turpin, RMIT University, Australia
Alexandra Uitdenbogerd, RMIT University, Australia
Geoff West, Curtin University of Technology, Australia
Hua Wang University of Southern Queensland, 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
Kyle Chard
Kathy Land
Jiangang Ma
Ben Palmer
Stephen Seidman
Xiaoxun Sun
Ian Welch
Guandong Xu

Organising Committee

Welcome

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

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

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

ACSW 2008 includes the following conferences:

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

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

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

General Chair

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

Organising Committee Members

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

CORE - Computing Research and Education

CORE welcomes all delegates to ACSW2008 in Wollongong.

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

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

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

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

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

CORE Computing Research & Education

Jenny Edwards
President, Computing Research and Education
January, 2008

ACSW Conferences and the Australian Computer Science Communications

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

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

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

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

Conference Acronyms

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

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 2008 Sponsors

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

University of Wollongong, Australia

AUSTRALIAN
COMPUTER
SOCIETY

Australian Computer Society

CORE - Computing Research and Education

THE UNIVERSITY OF AUCKLAND
NEW ZEALAND

Department of Computer Science

Department of Computing

Keynote

Constraint Logic Programming for Program Analysis

Joxan Jaffar

Department of Computer Science,
National University of Singapore
Email: joxan@comp.nus.edu.sg

Abstract

Constraint Logic Programming (CLP) has been traditionally applied to the modelling of complex problems, especially combinatorial problems, and to model knowledge bases. In this presentation, we focus on using CLP for program analysis and verification. First we consider the representation of program behavior: the rules and constraints of CLP provides for a natural specification of programs as a symbolic guarded transition system. The CLP execution model can then capture the symbolic traces of the underlying program, and these traces, in turn, divulge the properties that we seek. Secondly, we use the CLP formalism for the formal specification of complex properties of data structures. Here the CLP execution model can be used as a theorem-prover to dispense with the proof obligations arising from the program and its specifications. The traditional CLP execution model, however, is not automatically practical for these purposes. We shall present two refinements to CLP: one for reducing the number of symbolic traces that are needed to prove a property, and one to efficiently deal with data structure properties.

Invited Papers

On Measuring Java Software

Ewan Tempero

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

Extended Abstract

Software metrics have a reputation in industry of not being very useful. I believe one reason for this is that for most metrics one important aspect of them is usually not provided, namely the "entity population model". In measurement theory, an entity population model defines the typical values for measurements from a metric for a given set of entities. Having these models is necessary in order to interpret the measurements. For example, without knowing the entity population model for the body temperature of humans we would not know that someone with a temperature of 40 degrees would be a cause for concern. In order for software metrics to be useful we need to have a good understanding of their entity population models.

In fact, we know very little about the entity population models for software metrics for anything but the simplest forms of measurements. We do have speculations, expectations, and even some theories as to what they should be, but there has been very little data published that can help us know which are correct and which are not. There are various reasons why we do not have this data. Often it is because we do not know how to measure something, reuse for example. Sometimes there is disagreement as to what to measure - there are more than 20 metrics for cohesion of object-oriented software for example. But it is also the case that we simply have not made a consistent and sustained attempt to make and report such measurements. The few empirical studies that do exist suffer from lacking sufficient detail to allow them to be reproduced, or are from such a small sample that little can be determined from them. This is the situation I and others are trying to change.

In this talk I will discuss my experience in measuring Java software. I have found that just measuring a large collection of software provides interesting insights as to the state of current software development. It seems that no matter what is measured, the results are usually interesting and sometimes surprising. I will present some of these results. I will also discuss the issues involved in doing this kind of research. One such issue is making measurements that are reproducible. To address this issue, I advocate basing software metrics research on the use of standard software corpora, that is, creating collections of software whose contents are well-defined. However creating such a corpus is not just a matter of downloading stuff off the 'net. I discuss some of the difficulties that arise in developing a corpus of open source Java software.

Informatics Olympiads: Challenges in Programming and Algorithm Design

Benjamin A. Burton

Department of Mathematics, SMGS
RMIT University,
GPO Box 2476V, Melbourne, VIC 3001,
Email: bab@debian.org

Abstract

The International Olympiad in Informatics is a worldwide contest for high school students, with a strong focus on creativity and ingenuity in algorithm design. Here we describe the activities in Australia that support and complement this contest, including a range of programming competitions, more accessible pen-and-paper competitions, and other enrichment and training activities. Sample problems are included, along with suggestions for becoming involved.

1 Introduction

The International Olympiad in Informatics (IOI) is a prestigious international competition for high school students in programming and algorithm design. Created in 1989 in Bulgaria under the leadership of Petar Kenderov, it now boasts delegations and guests from around 90 different countries.

Students sit the IOI on an individual basis, and are given ten hours to solve six problems. The contest is a programming competition, in the sense that students submit programs which are then run through a variety of test scenarios and judged accordingly. However, the difficulty lies not so much in the programming but rather the design of the underlying algorithms.

Australia first entered the IOI in 1992, and became a regular participant in 1999. With a growing support base from academics, teachers and ex-students, a rich national programme is developing to support and complement the IOI.

The primary focus of this paper is to introduce the various activities that form the Australian programme. Section 2 presents an overview of these activities. In Section 3 we focus in detail on written competitions, a more accessible alternative that involves multiple choice and short answer problems, and in Section 4 we return to a detailed discussion of programming competitions. Broader activities within the Asia-Pacific region are discussed in Section 5, and Section 6 closes with suggestions for how teachers and students can become involved.

2 The Australian Programme

Like its sister programme in mathematics, the Australian informatics olympiad programme currently runs under the auspices of the Australian Mathematics Trust. Although the initial motivation for this programme was the annual selection and training of the Australian IOI team, it has since grown to provide enrichment for a wider range of high school students nationwide. Activities include:

- *The Australian Informatics Competition (AIC):* This is the most widely accessible activity, since it involves no programming at all. Held annually in May, it offers students a range of multiple choice and short answer questions that encourage algorithmic thinking in puzzle-like settings. The AIC and some sample problems are discussed in detail in Section 3.

  The AIC first ran in 2005, and has grown to over 3000 participants in 2007.

- *The Australian Informatics Olympiad (AIO):* The AIO is a true programming contest, and acts as the first round of selection in working towards an IOI team. Although the problems are necessarily simple, many of them retain a focus on algorithm design even at this early stage. See Section 4 for details and sample problems.

  The AIO has run since 1998. Numbers in this contest are typically much lower, with around 80 participants in 2007.

- *The School of Excellence:* The top twelve entrants from the AIO are invited to a live-in "programming boot camp" at the Australian National University in December, where they are given ten days of lectures, labs, contests and other activities. The school is intensive, and students typically emerge exhausted but full of ideas.

- *Invitational Contests:* The participants from the School of Excellence are invited to sit additional contests in February and March, including the French-Australian contest discussed in Section 5. These contests push the standard closer to IOI level, and (unlike the AIO) do not shy away from "required knowledge" such as graph theory and dynamic programming.

- *The Team Selection School:* Based on the invitational contests, a final eight students are invited to a second training school at Macquarie University. Where the focus of the December school is on teaching new material, the focus of this April school is on using this material in creative and unusual ways to solve problems of IOI difficulty. At the end of this school a final team of four members is chosen to represent Australia at the coming IOI.

- *The International Olympiad:* The four team members are individually mentored for the following 3–4 months. In August they meet for a final short but intense training school, after which they head directly overseas for the IOI.

Online materials are available through the national training site all year round (`http://orac.amt.edu.au/aioc/train/`), and a series of books to complement these materials is currently under development.

3 Written Competitions

It was noted in the introduction that the AIO—an entry level programming contest—has extremely low participation each year. Whilst there may be many reasons behind this, it is highly probable that the following factors contribute:

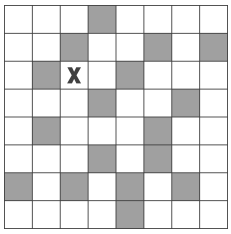
- Programming contests are difficult for schools to run. A typical mathematics contest requires nothing more than a desk and a pen. In contrast, a programming contest requires a computer for every student, appropriate software (compilers and debuggers), and a supervisor who can deal with technical problems if they arise.
- Programming contests require students who can write computer programs. In a mathematics contest, any student can follow their nose and scribble ideas down. In a programming contest—certainly the traditional type in which programs are scored according to their behaviour—a student cannot score any points (or even have their submissions judged) unless they can create a running program in a relatively short period of time.

For these reasons it was decided to complement the programming contests with a written contest, in the hope that this written contest might have broader appeal. The result was the Australian Informatics Competition, which has run annually since 2005.

Dungeon

(Australian Informatics Competition 2005, Intermediate)

A token (marked '`X`' in the diagram) is in a maze. You may move the token around according to the following rule: in each move the token may travel any distance either horizontally or vertically, but it cannot pass over or stop on a shaded square.



For example, from its starting position the token could travel either one square right, one square down, two squares down or three squares down in a single move. To reach any other square would require more than one move.

What is the minimum number of moves that you need to ensure that the token can reach any white square from its starting position?

(A) 8 (B) 9 (C) 10 (D) 11 (E) 12

Figure 1: The problem "Dungeon"

Although the AIC is styled as an informatics competition, AIC questions almost never use any code or pseudocode, and only a minority describe any explicit algorithm. Most problems pose some form of puzzle which, in order to be solved correctly, requires students to devise some type of informal algorithm in their heads.

An example from the first AIC is *Dungeon*, described in Figure 1. Although the problem can be solved by ad-hoc trials and guesses, it is faster and more reliable to work systematically outwards from the token, identifying all the squares that are one move away, then two moves away, and so on. Essentially the student who has never seen programming is encouraged to informally conduct a breadth-first search.

As well as multiple choice problems, the AIC contains a number of "algorithmic problems". An example is *Lost*, seen in Figure 2. Each algorithmic problem contains a task description followed by three scenarios, each of which can be solved with an integer in the range 0–999. The first scenario is typically small and easy to solve in an ad-hoc fashion, whereas the third is typically large and requires a systematic algorithm to solve quickly. The hope is that, as students attempt the simpler scenarios, they develop a feel for the problem and a systematic method that will allow them to tackle the larger cases.

Lost

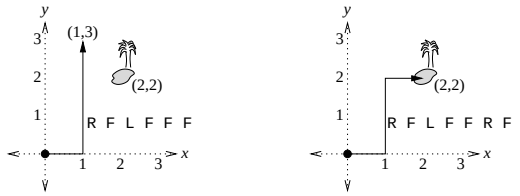
(Australian Informatics Competition 2007, Intermediate)

You are wandering through the desert with a map, which shows the desert as an (x, y) coordinate plane. You begin your journey at $(0, 0)$ facing north. In your hands are directions to an oasis, written as a sequence of letters. The possible letters are:

- `F`, indicating that you should walk forwards one kilometre in the direction you are currently facing;
- `L`, indicating that you should turn 90° to the left;
- `R`, indicating that you should turn 90° to the right.

Alas, the directions contain a critical mistake—one of the right hand turns has been deleted. Fortunately your map also shows the coordinates of the oasis, and so you hope to use this information to work out where the missing right hand turn should be.

For example, suppose the directions are `R F L F F F` and the oasis is at $(2, 2)$. The first diagram below illustrates this path, which ends at the incorrect location $(1, 3)$.



With some thought it can be seen that the directions should be `R F L F F R F`. That is, the missing right hand turn takes place just before the final walk forwards, as shown in the second diagram above.

Each scenario below lists a series of directions, followed by the location of the oasis. For each scenario, how many letters appear before the missing `R` must be inserted?

1. `RFFLFLFFFRFF` ⟶ $(3, 3)$

2. `RFFLFRFFLFFLFLFRFFR`
`FFLFLFRFRFFFRFLFFLF` ⟶ $(5, 5)$

3. `RFFFLFFRFFFFRFRFFRFFFF`
`FFLFFLFFFLFLFFFFFLFF` ⟶ $(8, 8)$

Figure 2: The problem "Lost"

Although written contests in computer science are relatively rare in comparison to programming contests, examples can be certainly be found elsewhere. One prominent example is the Lithuanian *Beaver* contest, which like the AIC encourages algorithmic thinking without explicitly requiring an understanding of computer programming (Dagienė 2006).

For a more detailed discussion of the AIC and additional sample problems, the reader is referred to Clark (2006).

4 Programming Competitions

Whilst written competitions can offer a highly accessible introduction to algorithms, the core activities of the Australian olympiad programme revolve around programming competitions.

The competitions offered in the Australian programme typically follow the model of the international olympiad. Students are given a large amount of time to solve a small number of tasks, each of which requires them to write a computer program. Each task describes the precise problem to solve, offers a simple text format for reading input scenarios and writing corresponding solutions, and sets time and/or memory limits within which the program must run.

A typical task of this type is *Mansion*, illustrated in Figure 3. Problems in the international olympiad are of course more difficult; worked examples are discussed by Horváth et al. (2002) and Burton (2007), and a comprehensive list of past IOI problems can be found at the IOI secretariat (`http://olympiads.win.tue.nl/ioi/`).

Readers might be familiar with tasks of this type from university programming contests, such as the ACM International Collegiate Programming Contest or the TopCoder contests. The International Olympiad in Informatics differs from these contests in the following ways:

- IOI tasks are graded on a sliding scale from 0 to 100, instead of an all-or-nothing pass or fail. This allows a range of scores for solutions of varying sophistication and efficiency.
- IOI tasks are extremely difficult to solve completely, since the running time and memory constraints for programs are often very tight. Whilst it might be straightforward to write a correct but inefficient solution that scores partial marks, it is often a significant achievement to score full marks for an IOI task.
- Students do not race each other. What matters is not when they submit each program, but only how it performs. This encourages stronger students to take their time in implementing sophisticated algorithms, in the hope of passing even the most difficult test scenarios.

Of course different styles of contest have different strengths. For instance, the all-or-nothing scoring for the ACM and TopCoder contests places a strong emphasis on rigour, whereas the extreme running time and memory constraints of IOI problems place the focus squarely on creative algorithm design. A more detailed comparison of different programming contests is given by Cormack et al. (2006).

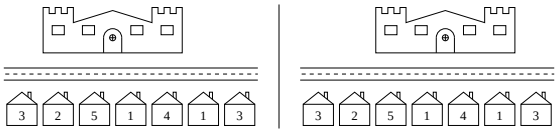
At the national level, the Australian Informatics Olympiad is intended as an entry-level programming contest for students with little or no formal training. At this level the scores are more likely to reflect the nuts and bolts of computer programming; as mentioned in Section 3, merely asking for correct running code is a relatively high bar for entry at the high school level.

Mansion

(Australian Informatics Olympiad 2007, Intermediate Q2)

You wish to build a mansion beside a long road. The far side of the road is filled with n houses, each containing a given number of people. Your mansion is as long as w houses combined. Your task is to position the mansion so that as many people as possible live across the road from you.

For instance, consider the road illustrated below, with $n = 7$ and $w = 4$. Here the seven houses contain 3, 2, 5, 1, 4, 1 and 3 people respectively. The first diagram places the mansion across from $2+5+1+4 = 12$ people, whereas the second diagram places it across from $5+1+4+1 = 11$ people. Indeed, of all the possible locations for the mansion, the largest possible number of people living across the road is 12.



Input: Your program must read its input from the file `manin.txt`. The first line of this file will give the integers n and w, and the following n lines will give the number of people living in each house.

Output: Your program must write its output to the file `manout.txt`. This file must give the largest possible number of people living across from the mansion.

Limits: Your program must run within 1 second. The input integers are guaranteed to lie in the range $1 \leq w \leq n \leq 100\,000$.

Sample Input and Output: The sample input and output files below correspond to the example given above.

`manin.txt`:

```
7 4
3
2
5
1
4
1
3
```

`manout.txt`:

```
12
```

Figure 3: The problem "Mansion"

Nevertheless, most problems in the AIO retain a focus on algorithm design. The challenge for the problem setters is to find *accessible* problems, where (i) the "obvious" algorithm is not necessarily the best, but (ii) good students with no formal training can be expected to find optimal algorithms through insight and creative thinking.

The problem *Mansion* (Figure 3) offers an example from the 2007 AIO. Essentially the problems asks, given an array of length n, for the continuous sub-array of length w with the largest possible sum.

A simple algorithm might loop through all possible starting points, and for each starting point loop again to sum the w elements of the sub-array. Whilst correct, this algorithm runs in quadratic time and is too slow to score 100% (in the AIO such a solution scored 70%).

This algorithm can be improved as follows. We retain the outer loop through all possible starting points, but we avoid the inner loop by using a *sliding window*. As we move from one starting point to the next, we adjust the sum by subtracting the one element that has been lost and adding the one element that has been gained. The resulting algorithm runs

in linear time, and is fast enough to score 100%.

It is pleasing to note that, of the AIO entrants who obtained correct or almost correct solutions to this problem, 18 used the linear algorithm and 21 used the quadratic algorithm. This suggests that the optimal solution was indeed non-obvious but nevertheless accessible, as the problem setters had hoped.

Figure 4 describes *Restaurants*, a more difficult problem from the 2007 AIO. Whereas the challenge in Mansion is efficiency, the challenge in Restaurants is correctness.

Restaurants

(Australian Informatics Olympiad 2007, Senior Q3)

You are faced with the unenviable task of organising dinner for an international conference. Several countries are represented at the conference, each with a given number of delegates. You have also identified several restaurants in the neighbourhood, each with a different number of seats.

In order to break down international barriers, you cannot seat two people from the same country at the same restaurant. Your task is to find an arrangement that seats as many people as possible.

As an example. suppose there are three countries with 4, 3 and 3 delegates respectively, and three restaurants with 5, 2 and 3 seats respectively. You can seat most of the delegates by placing one delegate from the second country and one delegate from the third country in every restaurant, and by placing two delegates from the first country in the first and third restaurants. This leaves two delegates without dinner, which is the best you can do.

Input: Your program must read its input from the file `restin.txt`. The first line of this file will give the number of countries, and the second line will give the number of delegates from each country. Likewise, the third line will give the number of restaurants, and the fourth (and final) line will give the number of seats in each restaurant.

Output: Your program must write its output to the file `restout.txt`. This file must give the smallest possible number of delegates who cannot be seated.

Limits: Your program must run within 1 second. There will be at most 5000 countries and 5000 restaurants.

Sample Input and Output: The sample input and output files below correspond to the example described above.

`restin.txt`:

```
3
4 3 3
3
5 2 3
```

`restout.txt`:

```
2
```

Figure 4: The problem "Restaurants"

Most of the students who attempted this problem adopted some type of greedy approach, and indeed the official solution is greedy (for each country, seat its delegates in order from the restaurant with the most empty seats to the restaurant with the fewest).

The difficulty is that not all greedy approaches are correct. For instance, some students adopted a similar approach but began with the largest restaurant instead of the emptiest; this works with the sample input and output, but does not work for more complex scenarios. In the end, 16 of the 27 students who attempted this problem scored 100%.

It is worth pausing to consider the ways in which this and other programming contests are judged. In particular, because solutions are judged entirely by their behaviour, students with partial or buggy implementations can score zero, even if they have derived the correct algorithm. In some cases (particularly in the IOI), good students may deliberately choose to submit an inefficient solution to avoid the risk of bugs that comes with more complex code.

This style of judging also raises pedagogical issues. Judging purely by behaviour does little to encourage good programming habits, and does not develop the communication skills that are crucial for teamwork and research in later life. This latter issue is explicitly addressed at the Australian training schools, where participants regularly present their algorithms to the other students and analyse them in a group setting.

The limitations of the current judging style are well understood, and the international community is actively engaged in finding ways to address them. Cormack et al. (2006) and Opmanis (2006) discuss the issues in depth and offer some concrete suggestions for improvement, and Burton (2007) examines them in the context of human-evaluated mathematics competitions. The IOI itself is actively evolving to find the right balance between competition, education and encouragement.

5 Regional Activities

To complement the national programme, it is valuable for students to engage in international competition and cooperation. Not only does this give them stronger experience in competition, but it also enhances the sense of camaraderie and helps them feel part of a wider community.

The IOI itself is a pinnacle of international competition, but with teams of four it can only be offered to a handful of students. To complement this, the regional and international communities have developed several smaller events that allow a greater depth of students to participate.

The first such event to appear on the Australian calendar was the French-Australian Regional Informatics Olympiad (`http://www.fario.org/`). This began in 2004 as a collaborative effort between Australia and France, and works well because the students of both countries have comparable skills. The contest has broader interest however, and each year a handful of students from other countries enter as unofficial participants.

More recently, the Asia-Pacific region has formed a new contest in the lead-up to the IOI. The inaugural Asia-Pacific Informatics Olympiad (`http://www.apio.olympiad.org/`) was hosted by Australia in 2007, with over 350 participants from 14 delegations. With Thailand and India lined up to host in 2008 and 2009, the contest is set to become a regular event on the regional calendar.

In addition to competitions, there is also collaboration in training between different countries. The Australian team met with the French in 2007 for a final week of joint training before the IOI, and two New Zealand students joined the Australians for the 2007 School of Excellence. At the teaching level, France and Australia regularly share problems and discuss training methods, and at IOI 2007 there was a mini-conference at which a diverse group of team leaders shared ideas and experiences.

6 Becoming Involved

For anyone eager to become involved in the programme as a teacher or a student, there are several excellent resources for learning more about programming contests.

Skiena et al. (2003) have written a superb book that focuses specifically on programming contests such as the IOI. It is very readable, contains a wealth of problems, and covers not only algorithms but also the practical issues of writing code in a contest environment.

Many countries have their own training sites, through which students can teach themselves in their own time. An excellent example is the USACO site (`http://www.usaco.org/`), which offers problems, reading notes and contest advice. The Australian site (`http://orac.amt.edu.au/aioc/train/`) includes all past AIO, French-Australian and Asia-Pacific papers. Both sites allow students to submit solutions with instant feedback, and are open to participants worldwide.

At the level of the IOI, Verhoeff et al. (2006) have proposed a "syllabus" of topics that might be covered. This list is currently under active discussion within the IOI community.

Interested people are also encouraged to contact their national organisation for information on local contests and training materials. The IOI secretariat (`http://olympiads.win.tue.nl/ioi/`) maintains a list of these organisations, alongside a wealth of archival material on the IOI and related competitions.

The Australian organisation can be reached through the author of this paper, who currently holds the role of Director of Training. The Australian Mathematics Trust, which oversees and administers the programme, can be reached through its Executive Director Peter Taylor at `pjt@olympiad.org`.

References

Burton, B. (2007), 'Informatics olympiads: Approaching mathematics through code', to appear in *Mathematics Competitions*.

Clark, D. (2006), 'The 2005 Australian Informatics Competition', *The Australian Mathematics Teacher* **62**(1) 30–35.

Cormack, G., Munro, I., Vasiga, T. & Kemkes, G. (2006), 'Structure, Scoring and Purpose of Computing Competitions', *Informatics in Education* **5**(1) 15–36.

Dagienė, V. (2006), 'Information technology contests — introduction to computer science in an attractive way', *Informatics in Education* **5**(1) 37–46.

Horváth, G. & Verhoeff, T. (2002), 'Finding the median under IOI conditions', *Informatics in Education* **1** 73–92.

Opmanis, M. (2006), 'Some Ways to Improve Olympiads in Informatics', *Informatics in Education* **5**(1) 113–124.

Skiena, S. S. & Revilla, M. A. (2003), *Programming challenges: The programming contest training manual*, Springer.

Verhoeff, T., Horváth, G., Diks, K. & Cormack, G. (2006), 'A proposal for an IOI syllabus', *Teaching Mathematics and Computer Science* **4**(1) 193–216.

Contributed Papers

A Model, Schema, and Interface for Metadata File Systems

Stijn Dekeyser **Richard Watson** **Lasse Motrøen**

University of Southern Queensland, Australia
{dekeyser,rwatson}@usq.edu.au, lassemot@yahoo.com

Abstract

Modern computer systems are based on the traditional hierarchical file system model, but typically contain large numbers of files with complex interrelationships. This traditional model is not capable of meeting the needs of current computer system users, who need to be able to store and retrieve files based on flexible criteria. A metadata file system can associate an extensive and rich set of data with a file, thus enabling more effective file organisation and retrieval than traditional file systems.

In this paper we review a wide range of existing proposals to add metadata to files and make that metadata available for searching. We then propose a hierarchy of definitions for metadata file systems based on the reviewed prototypes. We introduce a data model for a database-oriented pure MDFSs complete with operations and semantics. The model supports user-initiated instance and schema updates and file searches based on structured queries. We also explore the design space of a set of user interface operations intended to implement the pure model and facilitate the capturing of rich metadata. We argue that without such a simple method for users to create rich metadata, progress in this field will remain limited.

Keywords: Operating systems, Advanced applications of databases, Metadata.

1 Introduction

Traditional file systems store simple file metadata; a predefined set of data, mostly maintained by the operating system, is held in directories and file control blocks (e.g. inodes). Apart from assigning file names, users can effectively specify metadata by creating a directory hierarchy. The file path may encode some metadata. For instance the path `courses/csc2404/07/s2/ass1/1234/sync.c` assigns the following attributes to the file `sync.c`: *course=csc2404*, *year=2007*, *semester=2*, *studentId=1234*, *assignmentNum=1*, *filename=sync*, *filetype=Csource*. The ability to search based on attributes is limited as these attributes are stored hierarchically, and accessed via a path specification. It is a simple matter to build a search query that specifies all attributes in a file's path; this will yield all files in a directory. In our example, it is easy to locate all student assignment submissions for a particular offer of a course. However, a query that seeks to find all submissions for a particular student in a given semester is not supported.

To further explore these problems, consider the following common scenario. Bill has a multitude of music and image files on his personal computer and wants to organise his collection such that he can find and relate files easily. Using a traditional folder approach leads to various problems. The multimedia files can be placed in folders named according to several properties such as genre, year, band name, and location of photo. As discussed above, using hierarchical folders means that Bill loses the ability to search for files from different perspectives. He could populate the folders with soft links (or shortcuts) to the actual music files, but this would create an unacceptable burden of managing such links. Bill has installed third-party applications such as *Google Picasa* (for his image files) and *RealPlayer* (for his music files). These applications manage the organisation of files into groups based on the value of a property like "genre", which addresses the shortcoming of the folder approach. However it offers no solution if Bill wishes to link an image file to to a music file or if he wants to add his own metadata fields to a file.

This scenario demonstrates that organising multimedia using a traditional hierarchical file system, even when enhanced with specific applications, often proves to be impractical. The problem is not limited to multimedia as every type of file can have a large collection of metadata associated to it which can be used to organise the file space.

Problem Statement Simply stated, the first problem we address is that users must be able to manage files such that they can be located effectively at some future time. We need to be able to search for a file using multiple pathways (or search criteria). For example, we may use keywords that have been automatically extracted from the file, or attribute values (assigned by system or user), or links to related files, to seek the target file. A design for a metadata file system must include both the metadata storage model and appropriate user interfaces to allow a user to easily locate a file based on its metadata.

Critically, the second problem that we address is that a successful metadata file system must feature a user interface that allows users to easily assign meaningful and rich metadata. Requiring the user to create every piece of metadata through keyboard entry will almost certainly impede the adoption of such potentially revolutionary systems.

Existing Work Recently the advent of social networking websites that let users share images (e.g. *Flickr*) and video (e.g. *YouTube*) has demonstrated novel ways of organising multimedia. Such applications use the simple concept of *tags* to let users assign

metadata to their files, and allow others to search for files easily. On the users' own computers, more advanced applications such as Picasa and *Google Desktop* offer automated collection of metadata and use localised databases to store metadata and use it in search. Solutions proposed by researchers in the past decade took a more comprehensive approach by extending file systems with metadata functionality. On the commercial side, Microsoft is attempting[1] to implement a metadata file system called *WinFS*. We review these efforts in Section 2.

Contribution It is clear that various approaches to create, manage, and use metadata for files are being considered and developed, and that there is no single solution currently available that has wide adoption or satisfactorily solves all issues. In this paper we review a wide range of existing proposals to add metadata to files and make that metadata available for searching. We then propose a taxonomy for metadata file systems based on the reviewed prototypes. We introduce a data model for a database-oriented pure MDFS complete with operations and semantics. We explore a number of interesting and non-trivial issues that must be solved before a full-scale pure MDFS can be implemented. We also discuss two prototype implementations of our model and outline user interface interactions to capture rich metadata.

As evidenced by the fact that a major software company has not been able to deliver one after many years of work, it is clear that creating a truly useful and powerful MDFS is a daunting task. The problems are likely not only technical, but also of a more human nature (complexity for users, compatibility issues for businesses, etc). We therefore present our work as a modest step and as a basis for future extensions.

Note that the work presented in this paper is, within the context of computer science, of a highly multidisciplinary nature, drawing on results from multimedia systems, databases, programming languages, file systems, and human-computer interfaces.

2 Review of Existing Proposals and Systems

In this section we present existing systems and research proposals that attempt to overcome some of the shortcomings that are present in traditional hierarchical file systems. They include file systems specifically designed to make use of metadata and applications that make use of existing file systems to organise files. Due to space restrictions we refer the reader to [15] for a more detailed review of these and other systems (e.g. *Nebula* [3], *Windows Media Player*, etc).

Google Desktop Google Desktop provides a set of features that allows users to search for content on their computers, based on file name and, for some file types, content as well. When Google Desktop is installed on a system, it automatically indexes files on the computer. The index of words extracted from file names and, where possible, file content are stored in a local database. When new files are added to the system, or files are modified, the index is updated. Google Desktop relies on keyword search rather than structured queries. A search will retrieve a list of documents which are to some extent relevant to the keywords entered by the user. Hence, both the user interface and the kind of results are similar to those in the Google web search engine. A significant limitation is that users are not able to modify any metadata associated with a file. This can only be done by altering the file itself which will result in re-indexation. *Windows Desktop Search* is a similar system, based on the research prototype *Stuff I've Seen* (SIS) [7].

MIT Semantic File System The MIT Semantic File System [11] is one of the first file systems to address the shortcomings of traditional tree structured file systems. The main aim of the MIT Semantic File System (SFS) is to allow users to access files based on file content, as well as accessing files by name.

MIT SFS is designed to be integrated into a tree structured file system and it does so through the concept of *virtual directories*. Each virtual directory is interpreted as a query and contains symbolic links to the actual files stored in the underlying file system.

In order for SFS to provide file access based on file content (to make use of virtual directories as queries) the content of a file needs to be extracted. SFS does this by associating each file type with a *transducer* program that will extract the relevant metadata from files in the system. Each file type will have a specific transducer, and each transducer will be specifically designed to extract desired attributes and values from a file type. For example, a transducer for an email file may extract attributes "To", "From" and "Subject". MIT SFS comes with a set of default transducers that can handle the most common file types, but users are also able to implement their own transducers. A transducer table is used to determine which transducer to use for a certain file type.

Gifford et al. [11] outline some of the shortcomings of MIT SFS. The first point mentioned is that of the query language that each virtual directory can be associated with. MIT SFS offers only a basic query language that prohibits users from using boolean operators (such as 'OR', 'AND', etc.) to specify their queries. Users are also unable to assign metadata to files manually. It is also recognised by [11] that a more expressive data model should be utilised, instead of relying on simple attribute-value pairs.

To some extent, the open-source project MoveMetaFS [16] is similar to MIT SFS. The MMFS allows users to associate a set of tags to a file. The tags can be queried in a simple manner. The project mainly focusses on user interface operations to tag files, using a similar, but more limited, technique as we coined in [6] and describe in more depth in Section 7.

Haystack The Haystack project [12] is mainly based on the argument that developers cannot predict the ways a user wants to utilise information. All users have different needs and preferences when it comes to accessing information. Users should be able to specify relationships between different information objects, how these relationships should be presented, and how information should be gathered. Haystack currently accomplishes the required flexibility by storing all data using RDF [13]. Metadata for information objects in Haystack is initially automatically captured when a file is added to the system. It does so by generating RDF data using an *extractor* similar to MIT's transducer. The user interface for Haystack also allows users to easily modify metadata associated with files.

Aside from significant performance issues, an important shortcoming of the Haystack system is the absence of an API that other applications may use. Users have to interact with Haystack using its own user interface.

WinFS WinFS, like Haystack, attempts to offer a file system that allows information objects to be dy-

[1] Both the history and future of WinFS is relatively opaque. Contrary to earlier plans, it has not been shipped with Microsoft's Windows Vista operating system.

namically related to other information objects. Objects in WinFS can range from files of various types to persons, meetings, locations, etc., and they are all treated as information objects. The WinFS data model offers a rich set of operations that lets applications create, modify and query information objects, and is implemented on a relational database back-end.

WinFS allows applications to modify metadata stored in WinFS along with the schema for each information object, but offers only limited functionality to end-users. We argue that this unnecessarily curtails the usefulness of the system.

A less comprehensive open source project similar to WinFS is GNOME Storage which also uses a relational database backend. However, Storage focuses more on end-user keyword search rather than applications formulating structured queries. For more information on Storage and other systems we again refer the reader to [15].

Graffiti Graffiti [14] is a distributed organisation layer that augments an existing file system to add user-defined metadata and provide sharing of metadata across users and hosts. Graffiti supports the association of a simple text string tag with either a file or a pair of files (a named link). Apart from a linking capability, this differs from the common tagging systems (e.g. Flickr) in that it is generic rather than application-specific. Command line and graphical interfaces are provided.

While its metadata structure is unsophisticated, Graffiti addresses the problem of sharing files and metadata with some success. File checksums are used to ensure that a file and its metadata can be synchronised across multiple platforms. See section 5.2 for more discussion.

Linking File System The Linking File System [1] (LiFS) is a prototype implemented on top of a Linux filesystem. It augments a traditional file system with user specified attributes on files, and links between pairs of files. Links also have an associated set of attributes. The attributes are key/value pairs. LiFS implements the concept of a file trigger, which is an executable file attribute, encoded as a pattern/action pair. When a file operation occurs that matches the pattern, the associated action is executed. This generic mechanism is similar to the MIT SFS's transducer concept and could be used to implement the automatic collection of metadata (see Section 4.3).

The LiFS approach addresses the requirements of storing arbitrary user metadata. However, the absence of a metadata schema is an obstacle to the creation of advanced user interfaces, and implementation of powerful search queries. LiFS does not accommodate either specialisation of metadata for related file types through inheritance, or the ability to create non-file objects that could be linked to files. While the LiFS model is expressible using a database style model like WinFS, or that described in this paper, the reverse is not true.

3 A Taxonomy of Metadata File Systems

The full review of existing proposal and systems [15] brings to light a number of features which can be used to classify the systems into categories, and assist in constructing a taxonomy for a variety of metadata file systems and applications.

Data Model The data model for the reviewed systems ranges from attribute-value pairs over a relational model to RDF graphs.

Metadata–Filesystem integration Some applications maintain metadata in special purpose databases, and offer no filesystem functionality (e.g. for launching files). More advanced systems integrate the storage of metadata within the file system itself, and also offer rich operations on files.

Metadata Capture and Modification Most systems implement the concept of the MIT SFS transducer to capture metadata automatically. Few systems allow users to modify the metadata manually. Hence, rich metadata that is impractical to capture automatically (e.g. appearance of persons in images) is often neglected (some applications will allow users to add information in a predefined "comments" field).

Metadata Schema Modification very few systems allow users (or even applications) to modify the schema of the metadata store. Hence it is often impossible to create new types, new attributes, or new relationships between types.

Dynamic Views Only a few systems support the concept of dynamic views of objects defined by metadata properties. In addition, the expressive power of the view definition language is very limited.

We now propose a hierarchical classification of metadata file systems and applications.

Definition 1 (Metadata Enabled Application). *A Metadata enabled application is a stand-alone software package that runs on top of a host file system and has the following properties:*

1. *Manages its own database of metadata for files of a limited number of types.*
2. *Has a user interface that allows files to be organised based on the metadata, and allows users to search for files using keywords or simple attribute-value comparison.*

Such applications typically lack the ability to relate files of different types and to modify the schema of the metadata store. Examples of such tools include Google Picasa, Windows Media Player, Graffiti, and even MIT SFS.

Definition 2 (Rich Metadata Applications). *A rich metadata application supports the features of a metadata enabled application and also runs on top of an existing file system. It has the following additional features:*

1. *Allows end-users full power to manage metadata previously captured automatically, and allows users to relate files of different types.*
2. *Allows the schema for the metadata store to be modified.*

Such applications typically lack an API that other applications can use, and are not well integrated with the host filesystem. Examples of rich metadata applications include Nebula and Haystack.

Definition 3 (Metadata File Systems). *A Metadata file system (*MDFS*) supports the features of a rich metadata application but is tightly integrated with the traditional features of a filesystem. In addition it uses an expressive data model (i.e. relational, object-relational, object-oriented, or semi-structured), and has a comprehensive API to be used by third-party applications.* Examples of such systems include Microsoft's WinFS and GNOME's Storage.

Finally we present the definition used in the remainder of this paper.

Definition 4 (Pure Metadata File Systems). *A Pure metadata file system is an* MDFS *built on an object-oriented data model and features a powerful generic graphical user interface allowing end-users to fully manage metadata and schema modification.*

WinFS is not a pure MDFS because it is aimed towards software developers rather than end-users. While Microsoft's policy has the advantage of simplifying implementation and has the potential of making the introduction of WinFS on the desktop more palatable, we take the view that it is unnecessarily restrictive and misses the opportunity to present end-users with a potentially revolutionary new approach to file management. Indeed, users of WinFS will need to rely on applications to capture and use metadata and especially on their provisions to associate files of various types. Hence, users will not have access to a generic file browser for this functionality.

4 A Model for a Pure MDFS

In this section we present the formal definition of a data model for a metadata file system. What form should such a model take? A model must support the association of named attributes to files, and also have the ability to record relationships between files. A metadata file system must also support non-file entities (e.g. Persons) in order to be able to store complex metadata. A key feature of a pure metadata file system is that users can extend the metadata structure, typically by specialising an existing entity. Specialisation can be implemented using inheritance. The features described so far correspond closely with the entity-relationship data model. However, we will also need to provide special behaviour to entities (see Section 4.3).

In essence then, the data model we use at the lowest level is almost a subset of the ODMG Object Model [17, 5]. A significant departure from the ODMG model is that the deletion of a class from the schema has novel and unusual semantics. As class deletion implies object deletion, removing a class in the usual manner could also remove files. We define a more sophisticated delete operation (section 4.2.1) that does not result in file deletion.

We support the functions of a metadata file system by defining a series of schemas over the data model. Such schemas naturally form a hierarchy, with the base level ignoring concrete issues such as built-in classes and relationships, an fundamental file attributes. Schemas at higher levels are defined through extending the base level with new classes and attributes, mostly through inheritance. The next level in the hierarchy shown in Figure 1, the minimal schema, provides generic classes, file attributes and relationships. It is the minimum system that could be employed by users of an operating system which includes an MDFS file system.

In many cases this vanilla file system will be extended by operating system vendors and distributors to meet their requirements, for example shipping standard multimedia and word processing document classes and built-in relationships between email and person classes. Organisations could further extend the metadata schema to meet corporate and project specific needs. Finally, individual users can add to the hierarchy of classes if needed.

The rich hierarchy of built-in classes and relationships in the schema that end users will obtain with their OS, means that a possible proliferation of mutually incompatible schemas is somewhat mitigated. In addition, any two MDFS volumes will share at least part of the class hierarchy, making data and schema integration issues less of a problem. We discuss this further in Section 5.

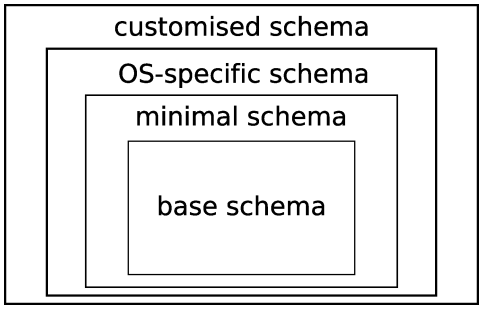


Figure 1: Filesystem schema levels

We will now define models for a schema and instance of a schema. The operations defined over the schema and instance are the application programming interface (API) to the metadata file system. Like their counterparts in a traditional file system, these would be implemented as system calls and execute in kernel space.

4.1 Data Model

The data model is class based, with relationships and simple inheritance. A schema defines classes and relationships between classes. Classes are also elements of an inheritance tree that is part of a schema. In the minimal schema we distinguish between classes that are associated with files (fileable) and those that are not (abstract). An instance over a schema defines instances of classes (objects) and relationship instances.

4.1.1 Names, identifiers, and values

The sets of possible class, relationship, attribute, and type names are respectively: *class*, *rel*, *att* and *type*. Objects and files are identified by members of the sets *oid* and *fid*, whereas *value* is the set of values. If $type = \{t_1 \dots t_n\}$ and $\mathrm{dom}\, t$ is the set of values associated with type t, then $value = \mathrm{dom}(t_1) \cup \mathrm{dom}(t_2) \cup \dots \mathrm{dom}(t_n)$. Each type contains a NULL value.

4.1.2 Schema definitions

A schema defines classes, organised into a specialisation hierarchy, and relationships between classes. We define the schema as the triple (class function, hierarchy relation, relationship relation). We define three supplementary functions on classes P (parent), S (superclass), and A (attribute), that are used in defining the schema operation semantics, and the instance semantics.

Each class introduces new attributes (class function C below), and instances of a class (objects) will contain the union of attributes defined by all ancestor classes. A class also inherits relationships from its ancestors.

$\mathcal{S}$ The *schema* $\mathcal{S} = (C, R, H)$.

C The *class* function $C : class \nrightarrow (att \nrightarrow type)$.

H The class *hierarchy* $H \subseteq class \times class$ such that H is the set of branches between classes, forming a single, rooted tree. Hence $(c_1, c_2) \in H$ means

that c_2 inherits directly from c_1, or c_1 is the parent class of c_2 (see P below).
$\forall (c_1, c_2) \in H \bullet \mathrm{dom}(C(c_2)) \cap \mathrm{dom}(A(c_1) = \emptyset$.
That is, subclasses can only extend a superclass definition.

A few classes and a small hierarchy is predefined in the minimal schema (the base schema is empty). There is a single root class (T_C), which has two subclasses (F_C and A_C). All user-created classes in the schema inherit from one of these two subclasses.

T_C The root of H. $T_C \in \mathit{class}$.
$C(T_C) = \{\mathit{creationTime} \mapsto \mathit{time}\}$

F_C The 'fileable' superclass $F_C \in \mathit{class}$.
$(T_C, F_C) \in H$.
$C(F_C) = \{\mathit{fileId} \mapsto \mathit{fid}\}$

A_C The 'abstract' superclass $A_C \in \mathit{class}$.
$(T_C, A_C) \in H \wedge T_C \neq F_C$

R The set of *relationships* $R \subseteq \mathit{rel} \times \mathit{class} \times \mathit{class}$.
A class inherits its ancestors' relationships, so the same relationship cannot be defined for the descendant classes.
$\forall (r, c_1, c_2) \in R \bullet \neg \exists (r_1, c_3, c_4) \in R \bullet r = r_1 \wedge c_3 \in S(c_1) \wedge c_4 \in S(c_2)$.

P The *parent* function on classes $P : \mathit{class} \nrightarrow \mathit{class}$.
$P = H^{-1}$.

S The *superclass* function $S : \mathit{class} \rightarrow \mathbb{P}\, \mathit{class}$.
$S = P^{+}$.

A The *attribute* function on classes.
$A : \mathit{class} \nrightarrow (\mathit{att} \nrightarrow \mathit{type})$.
$A(c) = \{(a, t) \mid c' \in S(c) \wedge (c'', f) \in C \wedge c' = c'' \wedge (a, t) \in f\}$. $A(c)$ defines the attributes, some inherited from superclasses, of an object instance of c.

4.1.3 Instance

The instance of a schema is modelled by a set of functions that encode the state of objects (O), and provide a means of identifying objects (I_C), and relationship instances (I_R). We also present definitions for two functions on object identifiers that are used to reveal the type (T) of the associated class, and its attributes (A_O). These are used in defining the semantics of the instance operations.

$\mathcal{I}$ The instance of schema $\mathcal{S}$. $\mathcal{I} = (O, I_C, I_R)$.

O The object function $O : \mathit{oid} \nrightarrow (\mathit{att} \nrightarrow \mathit{value})$.
The familiar *object.attribute* field access notation can be used as a shorthand:
$\forall\, o : \mathit{oid}, a : \mathit{att} \bullet o.a = O(o)(a)$

I_C The *instance* function identifies objects that have been created as an instance of a class.
$I_C : \mathit{class} \nrightarrow \mathbb{P}\, \mathit{oid}$.
$\forall\, c_1, c_2 \in \mathit{class} \bullet c_1 \neq c_2 \Rightarrow I_C(c_1) \cap I_C(c_2) = \emptyset$.
(Object is instance of just one class.)
$\forall (c, s) \in I_C \bullet \forall\, o \in s \bullet \mathrm{dom}(A(c)) = \mathrm{dom}(O(o))$
$\forall (c, s) \in I_C \bullet \forall\, o \in s \bullet \forall\, a \in \mathrm{dom}(O(o)) \bullet O(o)(a) : A(c)(a)$
(A class instance contains precisely the attributes of its instantiating class.)

I_R The *relationship instance* function
$I_R : (\mathit{rel} \times \mathit{class} \times \mathit{class}) \nrightarrow \mathbb{P}(\mathit{oid} \times \mathit{oid})$.
$\forall (r, c_1, c_2) \in R \bullet \forall (o_1, o_2) \in I_R(r, c_1, c_2) \bullet c_1 \in T(o_1) \wedge c_2 \in T(o_2)$.

T The *type* function on object identifiers
$T : \mathit{oid} \nrightarrow \mathbb{P}\, C$.
$T = S \circ I_C^{-1}$
$T(o)$ is a set that includes the class c that was used to instantiate o as well as all the superclasses of c.

A_O The attribute function on object identifiers.
$A_O : \mathit{oid} \nrightarrow (\mathit{att} \nrightarrow \mathit{type})$.
$A_O = A \circ I_C^{-1}$.

4.2 Operations

The following operations are sufficient to maintain the schema and an instance of a metadata store. Their semantics are defined with respect to the data model.

For each operation we show state transformations, the return value and, if the operation is partial, the exception condition. A transformation of a relation X is typically described as $X' = f(X)$, where X is the state of X before the operation and X' the post-operation state.

4.2.1 Schema Operations

The following operations populate a schema. Note that the schema "delete" operations also affect the instance. The delete class operation is very powerful and has interesting semantics. It removes the nominated class and all subclasses, and any relationships that relate those classes. Class deletion removes instances of deleted relationships, but *objects are not deleted*. Objects are instead recast as instances of the parent of the class being deleted. This will result in deletion of some attribute values.
The initial state of the minimal schema is:

$$\mathcal{S} = \{\{\ T_C \mapsto \{\mathit{creationTime} \mapsto \mathit{time}\},\ F_C \mapsto \{\mathit{fileId} \mapsto \mathit{fid}\},\ A_C \mapsto \emptyset\ \},\ \emptyset,\ \{\ (T_C, F_C), (T_C, A_C)\}\}$$

createClass(c, p, m): $\mathit{class} \times \mathit{class} \times \mathbb{P}(\mathit{att} \times \mathit{type}) \rightarrow \mathit{bool}$
$C' = C \cup \{c \mapsto m\}$
$H' = H \cup \{(p, c)\}$
Returns: $c \notin \mathrm{dom}\, C \wedge p \in \mathrm{dom}\, C$

deleteClass(c): $\mathit{class} \rightarrow \mathit{bool}$
$C' = C \lhd D$
$H' = \{(p, c) \mid (p, c) \in H \wedge p \notin D \wedge c \notin d\}$
$R' = \{(r, c_1, c_2) \mid (r, c_1, c_2) \in R \wedge c_1 \notin D \wedge c_2 \notin D\}$
$O' = O \oplus \{(o, O(o) \lhd \mathrm{dom}\, A(P(c))) \mid c \in T(o)\}$
$I'_C = I_C \lhd D$
$I'_R = I_R \lhd \{(r, c_1, c_2) \mid (r, c_1, c_2) \in R \wedge (c_1 \in D \vee c_2 \in D)\}$
Returns: $c \in \mathrm{dom}\, C$
where $D = \{c\} \cup \{x \mid c \in S(x)\}$

createRelation(r, c_1, c_2): $\mathit{rel} \times \mathit{class} \times \mathit{class} \rightarrow \mathit{bool}$
$R' = R \cup \{(r, c_1, c_2)\}$
Returns: $(r, c_1, c_2) \notin R$

deleteRelation(r, c_1, c_2): $\mathit{rel} \times \mathit{class} \times \mathit{class} \rightarrow \mathit{bool}$
$R' = R \setminus \{(r, c_1, c_2)\}$
$I'_R = I_R \lhd \{(r, c_1, c_2)\}$
Returns: $(r, c_1, c_2) \in R$

4.2.2 Instance Operations

Operations that access the files, but do not update metadata, are not presented here. A small set of operations (*open*, *close*, *read*, *write*, and *position*) would be required; only *write* is likely to affect the contents

of the metadata store; we discuss this is section 4.3. Note also that the *createFile* and *deleteFile* operations defined below reflect only the metadata store semantics. *createFile* would create a zero length file in the file store, and *deleteFile* would remove a file from the file store. Initially, $\mathcal{I} = \{\emptyset, \emptyset, \emptyset\}$

createObject(c): $class \nrightarrow oid$
$O' = O \cup \{(o, \{(a, NULL) \mid a \in \mathrm{dom}(A(c))\})\}$ where $o \in oid \wedge o \notin \mathrm{dom}\, O$
Exception: $c \notin \mathrm{dom}\, C$
Returns: o
deleteObject(o): $oid \rightarrow bool$
$O' = O \lhd \{o\}$
$I'_R = \{(r, x_1, x_2) \mid (r, x_1, x_2) \in I_R \wedge o \neq x_1 \wedge o \neq x_2\}$
Returns: $o \in \mathrm{dom}\, O$
createFile(o): $oid \rightarrow bool$
Returns:
$F_C \in T(o) \wedge o.fileId = NULL \wedge setAtt(o, fileId, f)$
where $f \in fid \wedge \neg \exists\, o \bullet o.fileId = f$
deleteFile(o): $oid \rightarrow bool$
Returns: $F_C \in T(o) \wedge setAttr(o, fileId, NULL)$
setAtt(o, a, v): $oid \times att \times value \rightarrow bool$
$O' = O \oplus \{(o, O(o) \oplus \{(a, v)\})\}$
Returns: $o \in \mathrm{dom}\, O \wedge a \in \mathrm{dom}\, A_O(o) \wedge v : A_O(o)(a)$
getAtt(o, a): $oid \times att \nrightarrow value$
Exception: $o \notin \mathrm{dom}\, O \vee a \notin \mathrm{dom}\, O(o)$
Returns: $o.a$
relate(r, o_1, o_2): $rel \times oid \times oid \rightarrow bool$
$I'_R = I_R \cup \{((c_1, c_2, r), (o_1, o_2)) \mid (r_1, c_1, c_2) \in R \wedge r = r_1 \wedge c_1 : T(o_1) \wedge c_2 : T(o_2)\}$
Returns:
$\exists (r_1, c_1, c_2) \in R \bullet r = r_1 \wedge c_1 : T(o_1) \wedge c_2 : T(o_2)$
unrelate(r, o_1, o_2): $r \times oid \times oid \rightarrow bool$
$I'_R = I_R \setminus \{(r, o_1, o_2)\}$
Returns: $(r, o_1, o_2) \in I_R$
isClass(c): $class \rightarrow bool$
Returns: $c \in \mathrm{dom}\, C$
isAtt(c, a): $class \times att \rightarrow bool$
Returns: $a \in \mathrm{dom}\, A(c)$
isInstance(o, c): $oid \times class \rightarrow bool$
Returns: $c \in T(o)$
isRelated(r, o_1, o_2): $rel \times oid \times oid \rightarrow bool$
Returns:
$\exists (r_1, c_1, c_2) \in R \bullet c_1 \in T(o_1) \wedge c_2 \in T(o_2) \wedge r1 = r$
search(c, f): $class \times F \rightarrow \mathbb{P}\, oid$
Returns: $\{o \mid o \in isInstance(o, c) \wedge F\}$

F is a formula with the following syntax

$$\begin{array}{rcl} F & ::= & atom \mid F \wedge F \mid F \vee F \mid \neg F \\ & & \mid \exists\, o \bullet F \mid \forall\, o \bullet F \\ atom & ::= & isInstance(o, c) \mid isRelated(r, o, o) \\ & & \mid o.a\, \Theta\, o.a \mid o.a\, \Theta\, k \end{array}$$

where $c : class, a : att, k : value$ are constants, $\Theta : value \times value \rightarrow bool$ is a comparison operator and $o : oid$ is a object variable. All variables except o in a query expression must be bound, and expressions must be *safe*, meaning that they should not return infinite sets.

Example Suppose classes `Audio` and `Wedding` exist, as does a relationship `MusicUsedInWedding`. The following query returns all music by the band Abba listened to while attending a wedding in Oslo:

$search($`Audio`, o`.artist = 'Abba'`
$\wedge\, \exists\, w(isInstance(w,$ `Wedding`$)$
$\wedge\, w$`.location='Oslo'`
$\wedge\, isRelated($`MusicUsedInWedding`$, o, w))).$

4.3 Extending the Model with MDFS Transducers

In [15] we detail the extension of our model with an active component; due to space constraints we give an informal description.

Firstly, the model's class attributes are extended with *system* flags. The flag implies access restrictions that work in both the instance and the schema of the model. On the instance side, values of system attributes can be read but not modified by applications and end-users. Modification can only be done by the MDFS system as described further in this section. On the schema side, system attributes cannot be removed from classes, nor can they be modified.

Note that this is not a form of security (see Section 5), as access is not determined on the basis of a user's ID. Instead, this flag only denotes that some attributes are owned by the system and cannot be modified through the interface.

Secondly, classes are extended with behaviour in the form of MDFS *Transducers*. These are functions[2] that modify the values of an object's system attributes. The transducers are called each time an object's associated file stream is modified (hence also when the file is first created); to this end, the semantics of the file *write* operation mentioned in section 4.2.2 is modified.

Transducers automatically assign metadata that can (and indeed, should) be captured without user interaction. For example, keywords for a text document, the `from` field of email messages, or the dimensions of an image. A transducer recursively calls the transducer of its parent class, up to the top-most class T_C. This ensures that system attributes such as file size and access time are updated automatically, in this case by the transducers of F_C and T_C respectively.

Note that relationships between objects are not set by MDFS Transducers. Indeed, these constitute metadata that requires user interaction. Also note that if an application or end-user creates a new class that contains system attributes, a transducer for the class should also be supplied, otherwise the values of these attributes will always be set to NULL. Finally, transducers may (but are not required to) also set values of non-system attributes if they are currently NULL, but these may be overwritten by end-users.

5 Multiple Users, Multiple Volumes

The model defined in Section 4 is intended for a single user, single volume pure metadata file system. As Figure 2 shows, this basic (or personal) environment is but one of four possible combinations when the twin axes of number of users and number of volumes are considered. The ultimate goal of our research project is to define a metadata file system that allows multiple users and communication with other volumes, whether they be MDFS or other systems on the same computer or accessible over a network. In this section we briefly outline some of the issues involved.

5.1 Security Aware MDFS

When several users have access to the information stored in a single MDFS volume, the system will need to provide security. Traditional file systems employ a role-based security method [8] to control read and write access of files' contents. They also impose a very limited degree of security on metadata, in that they can make directories unreadable for some users, thereby hiding file names, sizes, etc. However, the main focus is on securing access to the file's bitstream.

[2] An MDFS transducer is a reserved class method similar to a *Constructor* or *Destructor* in Object-Oriented programming languages.

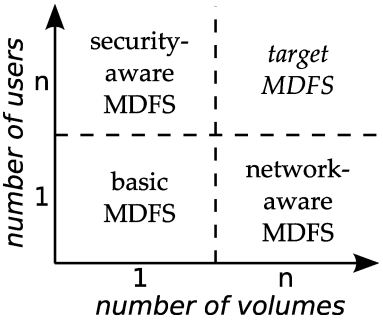


Figure 2: Extending the basic MDFS model to support multiple users and multiple volumes.

Interestingly, database systems also employ role-based security, but almost exclusively on the schema level. A database's INFORMATION_SCHEMA meta schema contains a table that stores users' privileges on such objects as tables and attributes. Databases do not offer security on the instance-level, where for example a user could be prohibited to read or write a tuple if the value of one of its attributes is a specific string.

For metadata file systems, we need role-based security mechanisms that work on the file contents level, the metadata level, and the schema level. It should also allow the use of instance-dependent access rules. For example, we may want to allow user A to write the content of file f_1 if the metadata field 'owner' is set to A's id, and the size of the file is less than 100 KB. A second user B may be allowed to read metadata associated with a file, but not to update it. Finally a user C may be allowed to create a new class by inheriting from an existing class in the MDFS schema, while other users are not.

We have studied instance-dependent access rules in the context of another research project [4] and aim to investigate their use in the context of metadata file systems.

5.2 Network Aware MDFS

There are several scenarios for the use of more than one file system volume. Consider that user A wants to copy a file from a remote network volume into his own MDFS volume. The remote volume may or may not be a MDFS system on its own, but will have some metadata associated to the file. In this scenario, where both the file content and metadata is copied to the local MDFS volume, the system, probably assisted to some degree by the user, needs to decide on a suitable mapping of metadata from the remote volume to the local volume.

Consider a second scenario, where two MDFS volumes exist and communicate via a network. Objects and files exist in one location, but both systems may use each other's objects and files. Clearly this scenario must solve the inherent schema integration problem [2].

In a third scenario user B may want to store all objects and files locally, but the metadata for an object is stored in a remote database. To make the example more concrete, suppose that a university stores data on its students in a central database, and individual lecturers have objects in their MDFSs for each of the students they teach. They can then create relationships between word processing documents and students in their local system. However, the student's metadata is accessed from the central database.

We note that Graffiti[14] is able to solve at least the problems of the first two scenarios, but only because of the simple and uniform tag metadata structure. The Graffiti server is able to synchronise files and metadata on two or more hosts because it does not have to deal with the issue of integrating file systems with different metadata structures.

These scenarios outline different solutions to the problem of letting a single MDFS volume communicate with other data systems. Ideally, a complete MDFS system will support each of the scenarios. We are currently investigating a range of solutions in this context.

6 Implementation Issues

6.1 Implementation Choices

Aside from storing files' binary content, a Metadata File System needs to store metadata and make it accessible through a query mechanism. The obvious target for implementing an MDFS is by using a database management system for the metadata, combined with a traditional file system to store file content. However, there are several possible avenues for implementing the model described in this paper. We briefly list some of them.

Employing Databases Since at the core our MDFS model is a subset of the ODMG Object Model, the preferred implementation platform is an ODMG-compliant object-oriented or object-relational database. Such a database is ideally equipped to handle class hierarchies and relationships. The actual binary content of individual files need not be stored in the database, however. It is sufficient to store a pointer to a file's inode in the database, and let the underlying file system handle subsequent file access. The database can natively handle queries, and also determine whether views (or *Virtual Folders*, as described in Section 7) are updatable. However, significant care must be taken in optimising the database and integrating it with the underlying file system.

Using WinFS Microsoft's WinFS was a data storage and management system based on a relational database. It is therefore similar to the previous option. We have reviewed the system in Section 2. The main advantage of using WinFS as an implementation base for a pure MDFS is that it natively supports many of the features that we require. However, the main disadvantage, apart from the incompleteness of the project, is that the metadata schema is not updatable at run time. This means that new classes must be compiled and made available as Dynamic-link Libraries. We argue that end users must be able to modify the schema (in particular the ability to create relationships) in order to achieve the full potential of metadata file systems.

Extending File System architecture It is possible to modify the way in which current file systems store metadata. For example, the POSIX inode structure could be extended to include a set of attribute-name/value pairs, and a set of pointers to other files. However, the notion of class with its specific attributes and its inheritance hierarchy would still be lacking, reducing users' power to model the metadata schema. On the other hand, query efficiency would be much less of an issue.

Use of links Soft links (or shortcuts) could be used to virtually place one file in various directories, the name of which represents a property of the file. As we

mentioned in the Introduction, managing the proliferation of links and directories would be a significant burden. In addition, relationships between files are difficult to represent in this manner.

Tagging File and directory names can be used to associate files with a set of tags. As mentioned in the Introduction, tagging is currently enjoying significant popularity in social web applications. Very likely this is due to the inherent simplicity of the method. Some of the metadata modelling power that we propose in this paper can be simulated by tagging, but concepts such as relationships pose a problem. In addition, querying and keeping tags consistent becomes rather convoluted.

6.2 Efficiency

This paper does not discuss efficiency issues as it concentrates on a model without bias towards any of the possible implementation platforms discussed in the previous section. However, it is clear that efficiency will be one of the deciding factors in the success or lack thereof of metadata file systems. We argue that current database technology is sufficiently far evolved to support the real-time data access needs required for this application. In addition, the LiFS [1] approach of storing metadata in new types of non-volatile main memory is a promising avenue of research.

6.3 Prototypes

Lasse At present we have finished work on a first tentative prototype of our model. The *Lasse* prototype was developed on top of a technology preview of Microsoft's WinFS. The main deliverable of *Lasse* was an MDFS File Browser application which allowed (1) the listing of objects in the MDFS file store, (2) a simplified mechanism to capture rich metadata (see Section 7), and (3) the creation of Virtual Folders (view definitions). The MDFS File Browser underwent a usability analysis by a number of staff in our department, which provided our project with crucial feedback to continue work on the model. The prototype also revealed the limitations of using WinFS as an implementation platform, mainly in the difficulty of letting users change the MDFS schema at run time. Screenshots of *Lasse* are included in Figures 3 and 4.

Sam We are currently working on a second proof-of-concept prototype, dubbed *Sam*. It is developed in Linux using the FUSE project which allows us to work in user-space and reuse common file browser components. We are using PostgreSQL as the database backend to store metadata, while file content itself will be stored in the default Linux file system. The goal of *Sam* is to further explore user interface issues; at first through a command-line interface, and later through a graphical layer.

7 User Interface Design Decisions

In Section 4.3 we introduced MDFS Transducers which, as in many commercial applications, handle the automatic capture of metadata without the need for user interaction. However, we argue that rich metadata such as links between objects cannot be captured automatically and requires user interaction. We also claim that without an efficient and effective generic GUI technique that helps end-users to rapidly capture rich metadata, MDFS technology will not be successful. In future work we will substantiate our claim by performing a usability study on a number of prototypes which we are implementing. In this section we briefly describe several aspects that have guided us in designing GUI operations for capturing metadata as well as using it in search. Again we refer the reader to [15] for more details.

Central in our approach is the concept of a virtual folder, or dynamic *View*. The visual presentation of a View is similar to a file browser (i.e. a table with one row per file and columns for metadata attributes) available in popular platforms, but Views have significantly different semantics. Files (or rather, fileable objects) may appear in more than one View, while being stored only once. A View is defined on the basis of a structured query (using the `search` function described in Section 4) and is refreshed each time an object (or set thereof) is "moved" in or out of the View.

The *move* operation has a new meaning compared to traditional file browsers, and plays a central role in our approach for capturing rich metadata through user interaction. Objects that are dragged into a View will acquire metadata that is needed for the objects to be in the result of the View's definition. If this process is successful the View, when refreshed, will include the new objects. Hence, the View is updatable. There are several cases in which the process can fail; (1) if the objects dragged into the View are of a different type than the View's definition, (2) if the View definition is an unupdatable query, and (3) if read-only attribute values must be changed in order for the objects to appear in the result.

Following [9] we allow more complex queries to be updatable than only those ranging over a single relation (class) and not using aggregation. In particular, the joins that we use in Views contain the equivalent of a `where exists` subquery and relate primary key (object identifiers) only. Consider the query expression given in Section 4.2.2 and the end-user wanting to drag a set of audio files into a View defined by that query. If there is more than one wedding in the system with `location = 'Oslo'`, the GUI will prompt the user for clarification. He may want to link the new audio files to just one such wedding (the GUI will display a list to choose from), or in some cases opt for associating the audio files with all weddings that took place in Oslo. As membership in the relationships is decidable at run time, such Views become updatable as well.

Figure 3 illustrates the use of Virtual Folders as a means to capture metadata through a move operation. The screenshots of the *Lasse* prototype show that initially four Photo objects were selected from the "Photos" Folder and subsequently dragged into the Virtual Folder "*Photos with Comments 'Family Holiday'*". The second screen then shows the content of the latter, and shows that the four objects have obtained the necessary metadata to belong in the Virtual Folder.

Views are persistent and can be organised into an arbitrary hierarchy by end-users but can also be organised automatically. Since all Views are subsets of the object store, a natural hierarchy based on set containment is implicit in the model. However, as View definitions are First-Order Logic expressions, View containment on the basis of their query definitions is undecidable. Hence, an automatic organisation of Folders will need to work on the instance level and reorganise the hierarchy each time the content of a View is changed. This is a potentially expensive operation that end-users should be able to switch off, but can be optimised by pruning parts of the hierarchy that have not changed.

A large number of other non-trivial issues are asso-

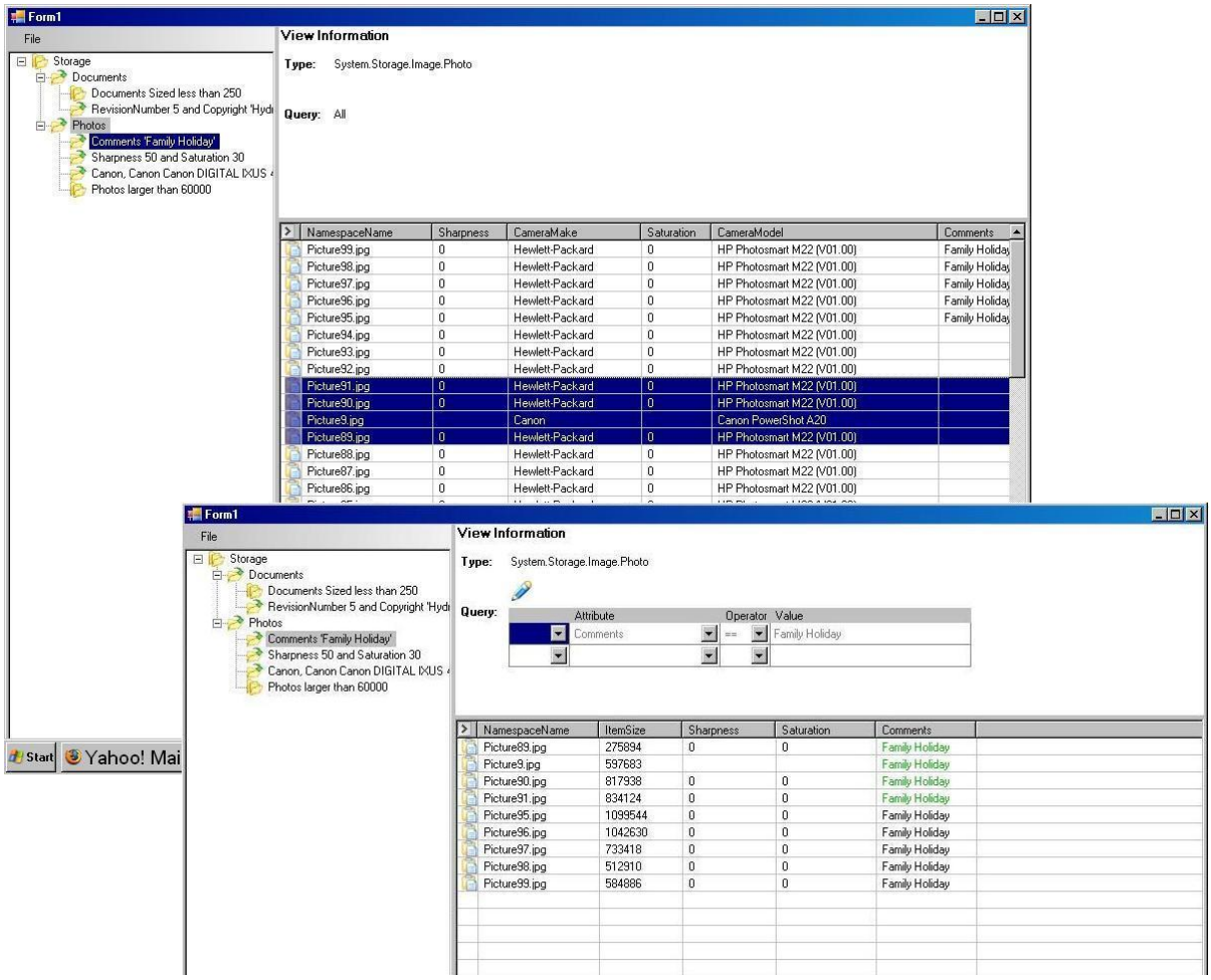


Figure 3: (a) Dragging photos into the Virtual Folder *Photos with comment 'Family Holiday'*, (b) Result after the drag operation, showing that metadata has been updated to make the photos appear in this Virtual Folder.

ciated with the operation of Views as a generic dual mechanism of querying the file store and acquiring new metadata for objects. For example, for a variety of reasons Views should return homogeneous sets of objects (as there always exists a common superclass for objects contained in a view), and it should be straightforward to decide which metadata attributes should be displayed to the user.

Deletion of Views should not result in the deletion of objects contained in the view, whereas the deletion of an object should also delete all relationship instances in which the object participated.

Creating Views should be possible in the GUI in a effective and simple manner. Users could be given a Wizard to create queries, use a Query-by-Example [19] interface (this is the approach we took in *Lasse*, see Figure 4 for a screenshot), employ a graph-based query language such as PaMaL [10], or orienteer [18] their way through class relationships to construct Views. Regardless of how views are created, users should see immediately whether their view is updatable. This facilitates the creation of a mental model so that users can employ the system more effectively.

Even when a very good structured query-definition interface exists, some users may opt to create a query consisting of key words. This could be either translated to a structured query, or information-retrieval algorithms can be used instead. Virtual Folders with an unstructured query definition can also be saved and placed in a hierarchy, but they are not updatable, and hence cannot be used to capture metadata.

These and other issues are detailed in [15].

8 Conclusion and Further Work

We have proposed a hierarchy of definitions for metadata applications and file systems based on a comprehensive review of existing implementations and research proposals. We then formally defined a model for a pure MDFS including data model, operations, and behaviour. Finally we described design aspects of a generic graphical user interface for capturing and using metadata in a pure MDFS. We are currently implementing a number of GUI prototypes, including those described here, and are planning a usability study with the aim to discover which strategies are most effective and efficient for capturing rich metadata through end-user interaction.

References

[1] Alexander Ames, Nikhil Bobb, Scott Brandt, Adam Hiatt, et al. Richer file system metadata using links and attributes. In *Proceedings of the 22nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST05)*. IEEE, 2005.

[2] C. Batini, M. Lenzerini, and S. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv.*, 18(4):323–364, 1986.

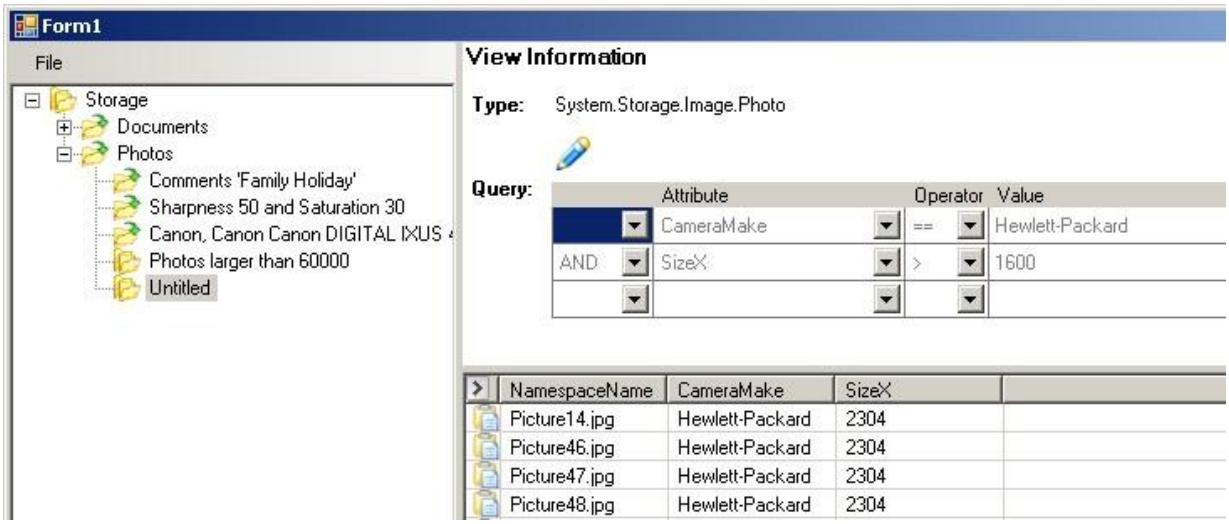


Figure 4: Creating a new Virtual Folder: query-by-example–like view definition interface.

[3] C. Mic Bowman, Chanda Dharap, Mrinal Baruda, Bill Camargo, and Sunil Potti. A file system for information management. In *Proceedings of the International Conference on Intelligent Information Management Systems, Washington D.C., USA, March 1994*, March 1994.

[4] T. Calders, S. Dekeyser, J. Hidders, and J. Paredaens. Analyzing workflows implied by instance-dependent access rules. In *ACM SIGMOD/PODS 2006 Conference*, Chicago, June 2006.

[5] R. Cattell, D. Barry, M. Berler, J. Eastman, et al. *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann, January 2000.

[6] S. Dekeyser. A metadata collection technique for documents in WinFS. In *Proceedings of the 10th Australasian Document Computing Symposium*. School of Information Technologies, University of Sydney, 2005.

[7] Susan T. Dumais, Edward Cutrell, Jonathan J. Cadiz, Gavin Jancke, Raman Sarin, and Daniel C. Robbins. Stuff I've Seen: a system for personal information retrieval and re-use. In *SIGIR*, pages 72–79. ACM, 2003.

[8] D. Ferraiolo and R. Kuhn. Role-based access controls. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[9] Antonio L. Furtado and Marco A. Casanova. Updating relational views. In *Query Processing in Database Systems*, pages 127–142. Springer, 1985.

[10] Marc Gemis and Jan Paredaens. An object-oriented pattern matching language. In *ISOTAS*, pages 339–355. Lecture Notes in Computer Science, Springer, 1993.

[11] David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, and James O'Toole. Semantic file systems. In *Proceedings of the Thirteenth ACM Symposium on Operating System Principles, Asilomar Conference Center, Pacific Grove, California, October 13-16, 1991*, pages 16–25. ACM, 1991.

[12] David R. Karger, Karun Bakshi, David Huynh, Dennis Quan, and Vineet Sinha. Haystack: A general-purpose information management tool for end users based on semistructured data. In *CIDR 2005, Second Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2005*, pages 13–26, 2005.

[13] Graham Klyne. Resource description framework (RDF), February 2004. W3C Recommendation.

[14] Carlos Maltzahn, Nikhil Bobb, Mark W. Storer, Damian Eads, Scott A. Brandt, and Ethan L. Miller. Graffiti: A framework for testing collaborative distributed metadata. In *Proceedings in Informatics*, number 21, pages 97-111, March 2007.

[15] Lasse Motrøen. Metadata file systems and GUI operations. Master's thesis, University of Southern Queensland, Australia, 2007. Draft.

[16] Szabó Péter. MoveMetaFS – a searchable filesystem metadata store for linux. Freshmeat project `http://freshmeat.net/projects/movemetafs`, 2007.

[17] Richard Soley and William Kent. The OMG Object Model. pages 18–41, 1995.

[18] Jaime Teevan, Christine Alvarado, Mark S. Ackerman, and David R. Karger. The perfect search engine is not enough: a study of orienteering behavior in directed search. In Elizabeth Dykstra-Erickson and Manfred Tscheligi, editors, *CHI*, pages 415–422. ACM, 2004.

[19] Moshé M. Zloof. Query-by-example: A data base language. *IBM Systems Journal*, 16(4):324–343, 1977.

Reasoning About Inherent Parallelism in Modern Object-Oriented Languages

Wayne Reid, Wayne Kelly and Andrew Craik

Faculty of Information Technology
Queensland University of Technology
2 George Street, GPO 2434, Brisbane QLD 4000, Australia

[wa.reid@student.,w.kelly@,andrew.craik@student.]qut.edu.au

Abstract

In the future, if we are to continue to expect improved application performance we will have to achieve it by exploiting course-grained hardware parallelism rather then simply relying on processor cycles getting faster. Programmers will, therefore, need to accept some of the burden of detecting and exploiting application level parallelism because automatic parallelization is still far from a reality. On the one hand we need to fundamentally reconsider how we express algorithms as the languages we currently use were never designed to make reasoning about parallelism easy. On the other hand, to be widely adopted, any new programming approach will need to be only incrementally different to current paradigms. This paper attempts to find that difficult balance. It extends modern object-oriented programming techniques with a new abstraction that allows either programmers or automatic parallelizing compilers to reason about inherent data parallelism.

Keywords: Reasoning; Parallelism; Object-oriented; Ownership types

1 Introduction

For decades, the IT industry has relied on faster processor clock cycles to deliver ever increasing application level performance. Fundamental laws of physics dictate that this will soon come to an end. Moore's law – "*that the number of transistors that can be placed on a chip will double every 18 months*" – is, however, forecast to continue in the foreseeable future. Therefore, there is the potential for application level performance to continue to improve, but these gains will be achieved through exploitation of course grained parallelism rather than simply relying on faster clock speeds. We have already seen the widespread release of dual and quad core chips and this trend is expected to continue. This poses an enormous challenge for the software industry; unlike faster clock speeds, application level parallelism can typically only be detected and exploited with some help from application developers.

The world of parallel programming has, until now, been a niche market characterized by high-end scientific applications executing on multi-million dollar supercomputers. Despite decades of research, the "holy grail" of automatic parallelization is still largely out of reach. Instead most resort to writing explicitly parallel applications involving message passing between processors. Only heroic efforts by highly trained scientific programmers have been able to exploit the massive parallelism provided by these supercomputers. Unfortunately, the languages used in these projects, like FORTRAN, are often decades old. The basic array data structures used in these older languages are easier to reason about than the complex pointer based structures of many more modern languages. The complexity of writing explicitly parallel programs decreases programmer productivity and increases defect rates in code compared to mainstream software development.

Expecting mainstream programmers to write explicitly parallel message-passing applications is clearly not reasonable. It is also unreasonable to ask them to fundamentally change the manner in which they express algorithms; for example, a switch to a purely functional paradigm would be too radical a change for most.

We cannot, however, simply ignore the pressing need to allow most applications developed in the future to exploit course grained hardware parallelism. If automatic parallelisation is too hard, we need to understand the fundamental cause of the difficulties. Rather than simply trying harder to overcome those difficulties, perhaps we should change the problem to side step the difficulties – failure is no longer an option.

Parallelizing a piece of code, either manually or automatically requires three logical steps: (1) decide if the code can be parallelized, (2) if it can, determine if any performance gain can be obtained, and (3) if it is worthwhile, refactor the program to include the constructs of parallelism (threads, semaphores, etc). We argue, in the context of modern general-purpose application development, that step (1) – the reasoning about parallelism – is the biggest challenge and is therefore the principal topic of this paper.

Over the decades, programming paradigms have evolved incrementally to include: structured control flow, abstract data structures, object-oriented programming, and component-oriented programming. Each evolutionary step basically places self-imposed limits on how we use certain programming language features; for example, the elimination of go-to statements. By restricting how we

program, we have improved comprehensibility and maintainability.

2 Reasoning about Parallelism

We argue that the next evolutionary step is to restrict the way that we program so that we can better reason about the inherent parallelism in applications. The fundamental challenge for both automatic parallelizing compilers and programmers trying to parallelize an application is being able to reason about what parallelism exists. It should come as no surprise that our current programming languages and paradigms are not particularly supportive of this goal since they were designed with sequential processors in mind.

Parallelism exists provided *dependences* do not exist. Dependencies are easy to define in terms of *runtime behaviour*: if one piece of code writes to some memory location that is later read by another piece of code, then there is a dependence between them and they cannot be executed in parallel. Reasoning about the presence of such dependencies is much harder to do statically. We have structured control flow and structured data, but the relationship between the code constructs and the data constructs is largely unconstrained. In modern object-oriented languages data and code are grouped into classes. The code in one class is free to mutate not just the data in its class, but also the data of other arbitrary classes. When trying to analyse the dependences in such code, abstract interfaces provide no useful information and we are forced to inspect the actual implementation of each method. This becomes very difficult in large complex applications and near impossible in the presence of components for which source code is not available.

The fundamental problem arises from the concept of mutable state; pure functional languages do not have such implicit dependencies. Rather than declaring mutable state and imperative programming to be "evil", we seek to learn how we can use it in a safe and controlled fashion. We seek to safely exploit its many benefits – just as our predecessors learned to safely use dangerous elements such as fire.

3 Abstracting Effect

What we need is a way of ***abstracting*** the ***effect*** of a piece of code (ie the set of memory locations that it reads and writes). The effect should, somehow, be in the interfaces to that code so that we can reason about its dependencies without examining its actual implementation. Object-oriented programming languages provide a good foundation to build on in this regard.

When trying to detect parallelism we are most interested in *scalable parallelism*: where the number of operations that can be performed in parallel is at least *O(n)* rather than *O(1)*. In these cases, the performance will generally continue to increase as we increase the number of processors with each new generation of multi-core chips. Scalable parallelism normally comes from parallelizing loops. A very important case of this is *data parallelism*: where some operation is performed in parallel on each element of some data collection. In an object-oriented language this would naturally take the form of a foreach loop iterating over the elements of some collection class invoking some method on it that performs the desired operation as shown in Figure 1.

```
foreach (T element in collection)
{
    element.Operation(arguments);
}
```

Figure 1: Stereotypical data parallel loop

This paper seeks to develop new abstractions to capture effects that allow conditions for parallelization to be easily reasoned about by both programmers and parallelizing compilers.

4 Sufficient Conditions for Parallelism

We start by informally considering what it would take for the loop in Figure 1 to be parallelizable. First, if the collection contains reference types then it is sufficient to ensure that there are no duplicate entries. Cohen, Wu, and Padua describe such collections as having the "*comb*" property, an analogy to the parallel teeth of a hair comb (2001). Note this property cannot generally be proved statically. It does not matter, however, what shape the collection takes; it may be a simple list, a tree, or a graph. The shape is not important provided the iterator visits each element exactly once.

Next we need to consider the properties of the Operation method itself. Basically, we need to guarantee that the method only mutates its own state and does not read the state of any other elements in the collection. But what do we mean when we say the state of an object? Clearly, it includes the fields of the object in question. If those fields contain pointers or references to other objects then, in many cases, those other objects should also be considered part of the state of the referencing object This encapsulation of internal implementation details is one of the hallmarks of object-oriented programming. Unfortunately, the "*private*" annotations used in most modern object-oriented languages do not ensure the kind of strong encapsulation we require. Marking a field or method as private means that "outsiders" cannot directly access that member, but there is nothing to prevent "outsiders" referring to the same object that the private field refers to. Such references might escape, for example, via a public method returning the value of such a private field.

5 Ownership Types

Other researchers have proposed stronger encapsulation mechanisms based on an idea called ownership types (Clarke et al., 1998). We borrow many of the concepts from that approach, so we take a moment to briefly summarize those concepts in this section.

Early versions of type systems to enforced encapsulation (Almeida, 1997, Hogg, 1991, Noble et al., 1998) were based on annotating some fields of an object as being part of its *representation*; this turned out to be insufficiently expressive. Consider, for example the use of a generic collection class such as a Stack within an object's

representation. In this scenario, the Stack class contains two types of fields, those that refer to data elements currently stored in the stack and those that form part of the internal implementation details of the Stack class itself. Clearly, the latter fields would be annotated as part of the *representation* of the Stack class. The former fields may not logically be part of the stack's internal representation, but they may logically be part of the internal representation of the object that contains the stack.

The idea, therefore, evolved to specifying the logical ***owner*** of each object. In the example just considered, the internal representation of the stack is owned by the stack, whereas the data elements contained in the stack are owned by the object containing the stack, or perhaps by some other object higher in the hierarchy.

The basic assumption is that all objects are owned by some other object. Globally visible top-level objects are said to be owned by a special context called *world.* It is assumed that the ownership relationship does not change during the lifetime of the objects.

These ownership relationships are represented by extending a language's static type system. Class definitions can be decorated with one or more ownership context parameters using syntax similar to generic types. In Figure 2, we declare a class named `Stack` with two formal context parameters named o and d.

```
class Stack[o, d] {
  ...
}
```

Figure 2: Declaration of a class with context parameters

By convention, the first context parameter (in this case *o*) always represents the owner of the current (or *this*) object. Other context parameters represent other owners possibly further up the representation hierarchy. Like generic types, these formal context parameters act as placeholders for actual contexts that must be provided when the class is instantiated. These actual contexts must be either *this*, *world* or a formal context parameter that is visible in the current scope.

Figure 3 shows an example of instantiating a Stack object; note that we provide the actual contexts `this` and `b` so that the context parameter *o* will be substituted by *this* and *d* by *b*.

```
class Sample[a, b] {
…
  public void op(…) {
    … = new Stack<this, b>();
    …
  }
…
}
```

Figure 3 Providing actual contexts when instantiating

The example in Figure 4 shows how a stack backed by a singly-linked list would be implemented when using ownership types.

```
class Stack[o, d] {
  Link<this,d> hd;

  Stack(Data<d> dt) {
    push(dt);
  }
  void push(Data<d> dt) {
    hd = new Link<this,d> (hd, dt);
  }
  Data<d> pop() {
    Link<this,d> ret = hd;
    hd = hd.next;
    return ret.data;
  }
}
class Link[o, d] {
  Link<o,d> next;
  Data<d> data;
  Link(Link<o,d>  next,  Data<d>  data)
  {
    this.next = next;
    this.data = data;
  }
}
```

Figure 4 Stack implementation with annotations

If a field is part of a class' representation then we annotate its type as being owned by `this`. Doing so effectively means that we cannot refer to that type from outside of the containing class. Outside the containing class the name *this* refers to a different class and there is no way of referring to the other *this*. If we cannot name a type, then we effectively cannot directly affect it nor can its internal references escape. This is the basic mechanism by which such ownership type systems provide stronger encapsulation.

A known limitation of the ownership types approach is that the encapsulation that it enforces is too strong to allow a number of common object-oriented design patterns such as factories and iterators. The idea was, therefore, extended by Lu and Potter to *effective ownership types* which weakens the conditions sufficiently to allow these common patterns to be expressed (2006). The key idea that we borrow from this work is the notion of a special existential context called *any*.

Ownership types (and effective owners) were originally designed to provide stronger encapsulation so that class invariants could be more easily reasoned about. They, therefore, included in their type systems various restrictions on which methods were allowed to read and write various objects based on their owner contexts. Whilst this is a laudable goal, we do not require any of those additional properties. For simplicity we do not include those additional restrictions in our type system. It is possible, however, for those additional restrictions to happily coexist with our type rules if the additional guarantees that ownership types provide are desired.

A nice feature that our system shares with the original ownership type systems is that it requires no runtime representation. Once the type checks are performed at

compile time, the ownership annotations can be completely dropped from the runtime type representations.

Both ownership types and effective ownership types require the programmer to provide the ownership type annotations. Deciding the logical owner of an object is a decision that requires human judgement. We feel that it is not an unreasonable question to ask a programmer or designer. Questions, like should this field be stored in a register or should this loop be parallelized, are arguably low level implementation decisions that should be handled by a compiler and not the programmer. The notion of ownership types allows designers to specify, in the code, high-level architecture decisions that should already be under consideration.

6 Annotating Effects

Now that we have a strong reliable notion of encapsulation, we return to abstracting the effect of our code constructs. The traditional approach to this problem is to calculate which fields are read and written by each method. This approach becomes increasingly complicated in modern object-oriented languages due to the complexity of programs written in these languages as well as the large number of aliases they contain (Lhoták and Hendren, 2006, Milanova et al., 2005). We, therefore, simplify and abstract the problem by summarising the sets of contexts that are read and written. These context sets must be specified in terms of contexts that are nameable in the current scope, so in some cases we need to raise contexts up to a context that contains them if the context actually affected is not nameable in the current scope.

Figure 5 shows a number of example methods and their corresponding read and write sets.

```
class A [o, d] {
  B<o,d> b = new B<o,d> ();
  void m1(C<d> c)
       write<o,a> read <this> {
     this.b.update();
     this.c.update();
  }
  void m2()
       write<this,o> read<this,world> {
     this.b = new B<o,d>();
     this.b.readWorld();
     this.b.c.e = …
  }
  void m3()
       write<this,o> read<this,world> {
     m2();
  }
}
```

```
class B[a, b] {
  … f = …
  C<a> c = new C<a>();
  void update()
       write<this> read<> {
     this.f = new…
  }
  void readWorld()
       write<> read<world> {
     …
  }
}

class C [a]{
  … e = …
  update()
       write<this> read<> {
     this.e = new…
  }
}
```

Figure 5: Calculating read and write context sets

We briefly discuss the effect calculation of class A's methods. Consider method m1 whose effects are a consequence of field and parameter modification. First we must compute the effect of b.update(). To invoke update on b we must first evaluate this.b which has the effect of reading the *this* context. We then lookup the effect of the update method in class B and see it writes to *this*; however, the *this* context referred to by update is not visible and so we must raise this effect to conclude that update writes to *o*, its owning context. Similarly, we then proceed to calculate the effect of this.c.update in a similar manner and conclude it reads *this* and writes to *d*. We can now conclude the effect of m1 is the summation of these effects, namely read *this* and write to *o* and *d*.

Next consider method m2 whose effects are the result of object creation, accessing *world* and the modification of a field via a chain of field accesses. The instantiation of B in m2 has no effect since the default constructor does not initialize anything. We write the reference of this new object to the b field in our class and so the overall effect is a write to *this*. To invoke readWorld on b, we must first read this.b which has the effect of read *this*. A lookup of the effects of readWorld then shows that it has the effect of read *world* which raises to itself. The net result of this statement is, therefore, a read of *this* and *world* with no write effects. Finally, the effect of writing to this.b.c.e is to read *this* (to obtain this.b) then to read *o* (to obtain this.b.c) and finally a write to the *this* context of class C which raises to *o*. The overall effect of the method is, therefore, write *this* and *o* and read *this* and *world*.

Finally, consider m3 which demonstrates the raising of effects within the same context. To compute the effects of m3 we lookup the effect of m2 and obtain write *this* and *o* and read *this* and *world*. Note that the *this* effects of m2 remain unchanged since m3 and m2 are in the same context which results in *this* being raised to *this*.

As will be shown in the next section, it is possible to automatically calculate the read and write sets of a method given its implementation. As a result, the programmer, therefore, can normally be spared the task of calculating read and write sets. There are however, a number of places where the read and write sets of methods need to be explicitly annotated rather than simply inferred. This occurs whenever we use any form of indirect interface. For example, if our language supported interfaces then we would need to declare the read and write effects of the methods in the interface and the methods of classes that implement that interface would need to limit their read and write sets to at most those defined in the interface. The same is true when a base class effectively serves as an interface by providing methods that can be overridden. Finally, read and write effects must be made an explicit part of component interfaces. If we need to use a class or component that is not annotated with context parameters or effects we simply assume that it reads and writes the *world* context.

7 Effect Computation

To formally demonstrate how effects can be automatically computed we present the abstract syntax and type rules which perform this computation. Note that the following rules do not constitute a complete type system; for clarity, we present only the effect calculation rules.

In the syntax of the following sub-sections, an over-line indicates a list.

7.1 Abstract Syntax

A program consists of a set of class definitions *L* and an expression to be evaluated e:

$$P \quad ::= \quad \overline{L}e$$

The definition for a class with name C consists of a set of context parameters $\overline{X}$, a subtype T, a set of fields f of type T and a set of method definitions M:

$$L \quad ::= \quad \text{class C}\left[\overline{X}\right] \lhd T\left\{\overline{T}\,\overline{f}; \overline{M}\right\}$$

A type T consists of the name of a class C and a list of actual context parameters $\overline{K}$:

$$T \quad ::= \quad \text{C}\left\langle\overline{K}\right\rangle$$

(the first context parameter is always the `owner` of T)

A context parameter K or E can be a real context (*this* or *world*), an existential context (*any*) or a yet to be bound formal context parameter X. I and J are context parameters with restricted domains.

$$\begin{array}{lll} K, E & ::= & X \mid this \mid world \mid any \mid unknown \\ I & ::= & X \mid this \mid world \mid any \\ J & ::= & X \mid this \mid world \end{array}$$

A definition for a method with name m consists of a return type T, a list of formal parameters x of some types T, and a body consisting of expression e. We also associate a set of read and write sets (I and J respectively) that can be automatically computed when we type the method body:

$$M ::= T\, m\left(\overline{T}\,\overline{x}\right) writes \left\langle\overline{J}\right\rangle reads \left\langle\overline{I}\right\rangle \{e\}$$

Expressions can be either a formal parameter, a new object instantiation, a field read or write or a method call:

$$\begin{array}{lll} e & ::= & x \\ & \mid & \text{new } T\left(\overline{e}\right) \\ & \mid & e.f \\ & \mid & e.f = e \\ & \mid & e.m\left(\overline{e}\right) \end{array}$$

7.2 Type Rules

We now present our formal type system for calculating read and write effects.

Type checking takes place within an environment Γ that maps formal parameters x to their corresponding type T.

$$\Gamma ::= x \rightarrow T$$

The primary goal of this section is to show how we can formally type and compute the read and write sets of arbitrary expressions. An expression e with read and write sets of I and J respectively and of type T evaluated under the environment Γ in the context K is represented as:

$$\Gamma; K \vdash e :^{\langle \overline{J}, \overline{I} \rangle} T$$

7.2.1 Formal Parameters

$$\frac{\Gamma(x) = T}{\Gamma; K \vdash x :^{\langle \varnothing, \varnothing \rangle} T}$$

Formal parameters are simply references to the actual objects. So, reading a formal parameter does not read or write the state of any objects or any ownership contexts.

7.2.2 Reading Fields

$$\frac{\varphi = \langle \varnothing, K \rangle}{\Gamma; K \vdash \text{this}.f :^{\varphi} \_}$$

The base case for this rule is reading a field from the current context. When we do this we generate no write effects and a read of the current context *this*. Note the _ in the rule is simply a placeholder for a value that is not used anywhere else in the rule and so is not assigned to a variable.

$$\frac{\Gamma; K \vdash e :^{\varphi'} T \qquad \varphi = \text{raise}(\varphi', K, T) \cup \langle \varnothing, \text{owner}(T) \rangle}{\Gamma; K \vdash e.f :^{\varphi} \_}$$

For the general case, we must first compute the type and effect of the expression e. The read set of the field access expression will include the owner context of expression e as well as the effects of computing the expression e. However, we must raise these effects up to a level of abstraction that can be named within the current class' context K. See section 7.3 for a definition of raise. Note that owner(*T*) is defined as:

$$\frac{}{owner\left(C\left\langle \overline{K}\right\rangle\right)= K_1}$$

7.2.3 Writing Fields

$$\frac{\Gamma; K \vdash e' :^{\left\langle \overline{J'},\overline{I'}\right\rangle} T' \qquad \varphi = \left\langle \begin{array}{c} \text{raise}\left(\overline{J'},K,T'\right)\cup\{K\}, \\ \text{raise}\left(\overline{I'},K,T'\right) \end{array} \right\rangle}{\Gamma; K \vdash \text{this}.f = e' :^{\varphi} T'}$$

In the base case of writing to a field in the current context, we must compute the type and effect of the right hand side of the assignment expression. We then raise the computed effects to the current context and add a write of the current context to the effects.

$$\frac{\Gamma; K \vdash e :^{\left\langle \overline{J},\overline{I}\right\rangle} T \qquad \Gamma; K \vdash e' :^{\left\langle \overline{J'},\overline{I'}\right\rangle} T' \qquad \varphi = \left\langle \begin{array}{c} \text{raise}\left(\overline{J},K,T\right)\cup\text{raise}\left(\overline{J'},K,T'\right)\cup\left\{\text{owner}\left(T\right)\right\}, \\ \text{raise}\left(\overline{I},K,T\right)\cup\text{raise}\left(\overline{I'},K,T'\right) \end{array} \right\rangle}{\Gamma; K \vdash e.f = e' :^{\varphi} T'}$$

We must first compute the type and effect of the expressions on the left and right hand sides. The read set of the assignment expression will include the owner context of the right hand expression e' and the raised read effects of e and e'. Similarly, the write set will include the owner of the left hand expression e and the raised write effects of e and e'.

7.2.4 Method Definition

$$\frac{\Gamma, \overline{x} : \overline{T}; K \vdash e :^{\left\langle \overline{J},\overline{I}\right\rangle} T \qquad \left(\begin{array}{c} \text{method}\left(\Gamma\left(super\right),m\right) = T\, m\left(\overline{T}\ \_\right)\text{writes}\left\langle \overline{J'}\right\rangle\text{reads}\left\langle \overline{I'}\right\rangle \\ \Rightarrow \vdash \overline{J} \preceq \overline{J'} \wedge \overline{I} \preceq \overline{I'} \end{array} \right)}{\Gamma; K \vdash T\, m\left(\overline{T}\,\overline{x}\right)\text{writes}\left\langle \overline{J}\right\rangle\text{reads}\left\langle \overline{I}\right\rangle\{e\}}$$

The advertised read and write effects of the method m must match the read and write effects of the expression e. If the method overrides a method in a super class then the advertised read and write effects of this method must be subsumed by the read and write effects of the super class method because the overriding method must be interchangeable with the method it replaces.

A context K is dominated by a context E (written $K \preceq E$) if K = E or K is owned either directly or indirectly by E.

7.2.5 Method Invocation

$$\frac{\begin{array}{c} \Gamma; K \vdash e :^{\varphi'} C\left\langle \overline{K}\right\rangle \qquad \Gamma; K \vdash \overline{e} :^{\varphi''} \overline{T} \\ \text{method}\left(T,m\right) = T' m\left(\overline{T}\ \_\right)\text{writes}\left\langle \overline{J}\right\rangle\text{reads}\left\langle \overline{I}\right\rangle \\ \text{class } C\left[\overline{X}\right]... \\ \varphi = \text{raise}\left(\varphi', K, C\left\langle \overline{K}\right\rangle\right)\cup\text{raise}\left(\varphi'', K, \overline{T}\right) \\ \cup\text{raise}\left(\left\langle\left[\overline{K}/\overline{X}\right]\overline{J},\left[\overline{K}/\overline{X}\right]\overline{I}\right\rangle, K, C\left\langle \overline{K}\right\rangle\right) \end{array}}{\Gamma; K \vdash e.m\left(\overline{e}\right) :^{\varphi} T'}$$

We must first compute the type and effect of the receiver object e. We then use that type to lookup the method m. We also need to compute the type and effect of the actual parameters. The net effect of the method invocation consists of the read and write effects of the method with actual context parameters substituted for the formal context parameters and the result raised to the current context, plus the raised effects of computing the receiver and actual method parameters.

Note that it is possible that the method *m* could modify an object passed by reference as an actual parameter. Because the types in our system contain the ownership information, the effect of modifying an actual parameter will be shown in the effect annotations of the method without any special handling.

7.2.6 Object Instantiation

$$\frac{\begin{array}{c} \Gamma; K \vdash \overline{e} :^{\varphi} \overline{T} \\ \text{method}\left(\text{C}\left\langle \overline{\text{K}}\right\rangle, C\right) = C\left(\overline{T}\ \_\right)\text{writes}\left\langle \overline{J}\right\rangle\text{reads}\left\langle \overline{I}\right\rangle \\ \text{class } C\left[\overline{X}\right]... \\ \varphi' = \text{raise}\left(\varphi, K, \overline{T}\right)\cup \textit{riase}\left(\left\langle\left[\overline{K}/\overline{X}\right]\overline{J},\left[\overline{K}/\overline{X}\right]\overline{I}\right\rangle, K, C\left\langle \overline{K}\right\rangle\right) \end{array}}{\Gamma; K \vdash \text{new C}\left\langle \overline{\text{K}}\right\rangle\left(\overline{e}\right) :^{\varphi'} \text{C}\left\langle \overline{\text{K}}\right\rangle}$$

Calling a constructor is largely the same as calling a method.

7.3 Raising Contexts

When we summarize the read and write effects of an expression we need to ensure that we use only contexts that are nameable from the current scope.

$\text{raise}(E, K, T)$ ::= the result of raising context E
from the scope of T into context K

We also overload the raise function to work over sets of contexts and pairs of read and write sets:

$$\frac{\varphi = \bigcup_{i=1}^{\left|\overline{E}\right|}\text{raise}\left(E_i, K, T\right)}{\text{raise}\left(\overline{E}, K, T\right) = \varphi}$$

$$\frac{\text{raise}\left(\overline{J}, K, T\right) = \overline{J'} \quad \text{raise}(\overline{I}, K, T) = \overline{I'}}{\text{raise}\left(\left\langle \overline{J}, \overline{I}\right\rangle, K, T\right) = \left\langle \overline{J'}, \overline{I'}\right\rangle}$$

Raising the *this* context requires us to determine if the *this* context being raised is the same as the current *this* context. If the *this* context is the same, then it is not abstracted by the raise function. Otherwise, we must abstract it to a visible context; that of its owner:

$$\frac{\Gamma; \_ \vdash K : T}{\text{raise}(\text{this}, K, T) = K}$$

$$\frac{\Gamma; \_ \vdash K : T' \quad T \neq T'}{\text{raise}(\text{this}, K, T) = \text{owner}(T)}$$

The *world* and *any* contexts are nameable form anywhere:

$$\frac{}{\text{raise}(world, \_, \_) = world}$$

$$\frac{}{\text{raise}(any, \_, \_) = any}$$

If the context to be raised is a formal context parameter, we must resolve the actual context to which the formal context parameter is bound:

$$\frac{\text{class } C\left[\overline{X}\right] \qquad \overline{X}_i = X}{\text{raise}\left(X, K, C\left\langle\overline{K}\right\rangle\right) = K_i}$$

8 Sufficient Conditions for Parallelism

With our abstraction for effects now defined, we return to developing the sufficient conditions for parallelization of our stereotypical data parallel loop:

```
foreach (T<c> e in collection)
      e.operation();
```

We want to ensure that the operation only mutates its own state. To ensure this, we will prove that it is sufficient to show that the operation's write set is either empty or `{this}`.

We also want to ensure that the operation does not read the state of any of the other objects in the collection. To ensure this, we will prove that it is sufficient to show that its read set is also either empty or `{this}`. This condition is sufficient but stronger than we would prefer as it also precludes the reading of other global state. Weakening this condition will be our primary goal in future work.

The following provides a formal proof that our stated conditions are sufficient to ensure safe parallelization.

We start by proving that if an expression (either directly or indirectly) writes to a field of an object, then either the owner context of that field, or a context which dominates that context will be included in the write set of the expression. We prove this by induction over our rules for computing write effects as defined in Section 7. The base case is the calculation of the write effects of a field assignment which directly inserts the owner of the field being modified into the expression's write set. The other type rules recursively ensure that if any writes occur as a side effect of those expressions, then either the owning context or the result of raising that context into the surrounding context is contained in the write set. The `raise` function, by construction, either returns the given context or a context which dominates the given context.

We assume that the read and write set of the operation are both `{this}` as the case of an empty read or write set is a more trivial special case (reading less or writing less can only help in proving our case).

We now show that if the operation's write set is `{this}` then executing `e.operation()` can (either directly or indirectly) only **write** to fields of objects that are either `e` itself or objects that are strictly dominated by `e`. Assume by way of contradiction that `e.operation()` does write to the fields of an object that is neither `e` nor strictly dominated by `e`. In this case, by our earlier proof for all expressions, either the owner context of the object written or a context which dominates it would be included in the method invocation's write set. But for that to be the case, according to our rule for calculating the write effects of a method invocation, that context would need to have come from the write set of either the receiving expression, the arguments or the method definition. In this case, there are no arguments, the receiver is simply a variable expression which has no side effects and the write set of the method is simply `{this}`. The `this` in this case refers to the expression `e`, so the method invocation only writes to `e` and contexts dominated by `e` - hence providing the contradiction.

Through the same reasoning process we can show that if the operation's read set is `{this}` then executing `e.operation()` can (either directly or indirectly) only **read** fields of objects that are either `e` itself or objects that are strictly dominated by `e`.

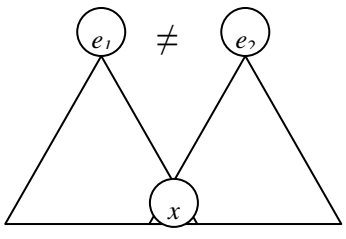
A loop can be parallelized unless a dependency exists between iterations of the loop. A dependence can take one of three forms:

Flow dependence: one iteration writes some memory location which is read in some later iteration.

Output dependence: one iteration writes some memory location which is overwritten by some later iteration.

Anti-dependence: one iteration reads some memory location before it is overwritten by some later iteration.

Assume by way of contradiction that a flow dependence exists. The collection must contain two elements e_1 and e_2 such that e_1`.operation()` writes to a field of some object x and *e2*`.operation()` reads that field of x.



We know from the earlier proof that x is either e_1 or owned by a context that is strictly dominated by $e1$. We also know that x is either e_2 or owned by a context that is strictly dominated by e_2. We know from the "*comb*" property of the collection that $e_1 \neq e_2$, so x is not e_1 or e_2. The object x must therefore be strictly dominated by both e_1 and e_2. Each object is only owned by one object. If x is strictly dominated by e_1 and e_2, it must be the case that either e_1 dominates e_2 or e_2 dominates e_1. But we know that e_1 and $e2$ are both directly owned by context c, which provides the contradiction (provided their owner c is not the existential context *any*).

As the read and write sets are actually the same, the same reasoning process can be used to rule out output and anti-dependences.

9 Parallelization

If the sufficient conditions are met then the loop can be parallelized. The performance benefit that can be obtained by parallelizing the loop is a separate question that depends, amongst other things, on the amount of work to be done per iteration. We consider this question to be outside the scope of this paper. If the decision is made to parallelize the loop then some kind of thread-like mechanism needs to be employed to enable the parallelism; this is relatively mechanical and again outside the scope of this paper. We do, however, consider what the programmer would need to provide to facilitate the loop parallelization. Help from the programmer may be required to convert the iterator used with the collection class into a parallel iterator. Ideally, the collection class is extended to provide both a sequential iterator and a parallel iterator.

A parallel iterator is simply a means by which to return multiple iterators, one for each thread of execution. Modern object-orientated languages typically define sequential iterator behaviour through the use of an interface. For example, the IEnumerator interface presented below with effective ownership annotations:

```
iterface IEnumerator[o, d]{
  void Next()
    writes<this> reads<this, any>;
  Data<d> element()
    reads<this, any>;
}
```

Similarly, collections that return iterators are typically required to implement an interface to advertise this capability. For example, the IEnumerable interface extended below with effective ownerships:

```
interface IEnumerable[o, d]{
  IEnumerator<o, d>
  GetIterator()
    reads<this>;
}
```

By extending the IEnumerable interface we can create a parallel iterator that returns multiple iterators as needed. An example follows:

```
interface   ParallelEnumerable[o,   d]:
IEnumerable[o, d]{
  IEnumerator<o, d>
  PIterator(int m, int N)
  reads<this>;
}
```

The `PIterator` method returns an iterator that contains a subset of the items in the collection. The value m is the processor and the value N is the total number of processors. The collection programmer must ensure that for each value of m the returned iterators contain mutually disjoint sets of elements and that an element be visited exactly once by only one of the returned iterators.

This approach allows the data structure designer to tailor the iterators returned to provide the most efficient traversal possible. Further, the number of iterators can be specified at compile or run time, thus insulating the iterator user from issues of load balancing and mapping.

On Processor m of N:

```
  foreach (T e in c.pIterator(m,N))
      e.Operation();
```

Collections are likely to exist in a set of standard class libraries so that most programmers would not have to worry about implementing them.

If there is no parallel iterator implementation, but a serial iterator exists, we can use an inspector-executor pattern (Ponnusamy et al., 1994). In the first sequential step, the sequential iterator is used to determine all the elements of the collection. Then the parallel step divides these objects up and iterates over them in parallel:

```
int count = 0
foreach (T e in c)
    temp[count++] = e;

int size = Math.Ceiling(count/N);
```

On Processor m of N:

```
int start = m * size;
int end = min(count, (m+1)*size-1);

for (int i=start; i<=end; i++)
    temp[i].Operation();
```

This approach would be used when access to the source code for the collection class is not available.

10 Related Work

As discussed earlier, parallelization requires three logical steps: reasoning about parallelism, deciding if parallelism is worth exploiting, and transforming the program into a parallel form. There is a huge amount of related work (Allen and Kennedy, 2002, Muchnick, 1997) on these latter two topics, but we do not discuss it here as our focus is on reasoning about parallelism.

The original work on ownership types, on which our system is based, was published by Clarke, Potter, and

Noble (Clarke et al., 1998). A number of different variations on this system have since been proposed to make standard design patterns easier to express and to enhance the information that can be derived from it. Lu and Potter extended ownership types to produce the effective ownership system that we extend (2006).

There are a few ownership type extensions which attempt to provide some of the same features as our system. We now list these systems and discuss their similarity to our contribution.

JOE extends ownership types to allow expression of common patterns as well as adding effects to allow for reasoning about the non-interference of code (Clarke and Drossopoulou, 2002). JOE's effect computations involve the language's dynamic semantics which we do not require making our system static as opposed to dynamic (Clarke and Drossopoulou, 2002).

Boyapati, Lee, and Rinard published a paper describing how pure ownership types could be used to reason about the safety of parallel code; more specifically guaranteeing the absence of data races and deadlocks (2002). Their system only reasons about the safety of locking protocols used within explicitly parallel programs rather than attempting to identify parallelism in a sequential program (Boyapati et al., 2002).

A few other languages have introduced effect notations and attempted to exploit the additional information they provide. The Jade programming language by Lam and Rinard allows programmers to annotate programs with unverifiable effects at the granularity of variable and relies on the programmer to reason about aliasing (1998).

Boyland, Noble and Retert presented several language features related to effect containment and ownership (Boyland et al., 2001). It does not explore concurrency or parallelism other than to state that encapsulation and abstraction are useful tools which potentially could be used to enable safer concurrent programming.

Many have tried to perform general purpose inter-procedural pointer alias analysis (Milanova et al., 2005, Lhoták and Hendren, 2006), but without any form of additional help from the programmer they tend to be very conservative as well as very expensive to perform.

There have also been efforts to build systems that employ formal logic to reason about inherent parallelism. Rus and Van Wyk employed temporal logic and model checking to identify opportunities for parallelism (1997). Dongol and Mooij used process logic to construct a system to derive safe concurrent programs from sequential ones (2006). These systems are very powerful, but they are also expensive to implement and the programmer has less control over their operation.

11 Conclusion

We have borrowed the notion of ownership types to create a new abstraction that allows both programmers and automatic parallelising compilers to reason about data parallel loops. An acknowledged weakness of our current approach is that our sufficient conditions for parallelization do not allow such loops to read any global state. Addressing this problem will be the principal focus of our future work. We have so far only considered data parallel loops in their simplest possible form, just one method call in the loop body and no arguments. We do not suggest that our techniques in their current form are ready to be applied to a wide range of real applications. It is, however, an important first step towards a new programming paradigm that is only incrementally different to current paradigms, yet capable of reasoning about parallelism. When faced with the extremely daunting challenge of trying to reason about parallelism, our philosophy has been that until we know how to address the simple cases, it is fruitless to consider the more general cases. This is, therefore, hopefully the first step of a long and fruitful journey.

12 References

ALLEN, R. & KENNEDY, K. (2002) Optimizing Compilers for Modern Architectures: A Dependance-Based Approach, San Francisco, CA, Morgan Kaufmann.

ALMEIDA, P. S. (1997) Balloon Types: Controlling Sharing of State in Data Types. ECOOP '97 - Object-Oriented Programming. Heidelberg, Berlin, Springer Berlin / Heidelberg.

BOYAPATI, C., LEE, R. & RINARD, M. (2002) Ownership types for safe programming: preventing data races and deadlocks. Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Seattle, Washington, USA, ACM Press.

BOYLAND, J., NOBLE, J. & RETERT, W. (2001) Capabilities for sharing: A generalisation of Uniqueness and Read-Only. 15th European Conference on Object-Oriented Programming (ECOOP). Budapest, Hungary, Springer Berlin / Heidleburg.

CLARKE, D. & DROSSOPOULOU, S. (2002) Ownership, encapsulation and the disjointness of type and effect. Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Seattle, Washington, USA, ACM Press.

CLARKE, D. G., POTTER, J. M. & NOBLE, J. (1998) Ownership types for flexible alias protection. Proceedings of the 13th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Vancouver, British Columbia, Canada, ACM Press.

COHEN, A., WU, P. & PADUA, D. (2001) Pointer Analysis for Monotonic Container Traversals (CSRD TR-1586). Univeristy of Illinois at Urbana-Champaign.

DONGOL, B. & MOOIJ, A. J. (2006) Progress in Deriving Concurrent Programs: Emphasizing the Role of Stable Guards. 8th International Conference on the Mathematics of Program Construction Kuressaare, Estonia, Springer Berlin / Heidelberg.

HOGG, J. (1991) Islands: aliasing protection in object-oriented languages. Conference proceedings on Object-oriented programming systems, languages, and applications. Phoenix, Arizona, United States, ACM Press.

LHOTÁK, O. & HENDREN, L. (2006) Context-Sensitive Points-to Analysis: Is It Worth It? 15th International Conference on Compiler Construction. Vienna, Austria, Springer.

LU, Y. & POTTER, J. (2006) Protecting representation with effect encapsulation. Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages. Charleston, South Carolina, USA, ACM Press.

MILANOVA, A., ROUNTEV, A. & RYDER, B. G. (2005) Parameterized object sensitivity for points-to analysis for Java. ACM Transactions on Software Engineering Methodology, 14, 1-41.

MUCHNICK, S. S. (1997) Advanced Compiler Design & Implementation, San Francisco, CA, Morgan Kaufmann.

NOBLE, J., VITEK, J. & POTTER, J. (1998) Flexible Alias Protection. IN JUL, E. (Ed.) Lecture Notes in Computer Science. Berlin, Hidleburg, New York, Springer Berlin / Hamburg.

PONNUSAMY, R., HWANG, Y.-S., DAS, R., SALTZ, J., CHOUDHARY, A. & FOX, G. (1994) Supporting irregular distributions in FORTRAN 90D/HPF compilers (CSTR3268.1). University of Maryland at College Park.

RINARD, M. C. & LAM, M. S. (1998) The design, implementation, and evaluation of Jade. ACM Trans. Program. Lang. Syst., 20, 483-545.

RUS, T. & VAN WYK, E. (1997) Model Checking as a Tool Used by Parallelizing Compilers. 2nd International Workshop on Formal Methods for Parallel Programming: Theory and Applications. Geneva, Switzerland.

Ruby.NET:
A Ruby Compiler for the Common Language Infrastructure

Wayne Kelly and John Gough
Faculty of Information Technology
Queensland University of Technology
2 George Street, GPO 2434, Brisbane QLD 4000, Australia
w.kelly@qut.edu.au, j.gough@qut.edu.au

Abstract

The implementation of statically typed programming languages on the .NET Common Language Infrastructure (CLI) is by now well understood (Gough 2002). However, the situation with dynamic languages is not so clear. Typically such languages have objects that are dynamically typed, while the CLI is statically typed at the instruction code level. Nevertheless there is a growing body of evidence suggesting that the CLI can be a suitable target for such languages (Hugunin 2006). In order to better understand the issues involved we set out to create a full implementation of the Ruby language on the CLI. This paper describes the challenges faced and design decisions made in creating Ruby.NET – a Ruby compiler for the CLI.

Keywords: Dynamic languages, Ruby, CLI, .NET and Compilers.

1 Introduction

This paper discusses the design of Ruby.NET – a Ruby compiler for the .NET Common Language Infrastructure (CLI). Ruby is "a dynamic, open source programming language with a focus on simplicity and productivity" (Ruby). The Ruby language has traditionally been implemented via an interpreter created by the language's inventor Yukihiro Matsumoto ("Matz"). The creation of a Ruby compiler for .NET adds the Ruby language to the stable of .NET languages and provides Ruby developers convenient access to the resources of the .NET platform.

We provide two front-ends to our compiler to better cater for the needs and preferences of developers from both the Ruby and .NET camps. Our `RubyCompiler` front-end compiles one or more Ruby source files into either `.dll` or `.exe` .NET assembly files. These assemblies can then be dynamically linked with other assemblies (possibly created using other .NET languages) to create an application. This front-end is designed primarily for developers coming from the .NET world and so takes approximately the same command line arguments as Microsoft's C# compiler. We also provide a Visual Studio integration package for Ruby.NET, for those wishing to develop Ruby projects within a common integrated development environment.

Our other front-end, `Ruby.exe` is intended to emulate Matz's Ruby Intepreter (MRI). It takes a single Ruby source file and executes it. Internally `Ruby.exe` compiles the Ruby source file into a .NET assembly and then loads and executes it, but this is all transparent to the user as no assembly files are actually written out to disk. This front-end is designed primarily for developers coming from the Ruby world and so takes approximately the same command line arguments as MRI.

These two front-ends are in fact thin wrappers around a common core called `RubyRuntime.dll` that contains the entire compiler infrastructure as well as an implementation of all of Ruby's built-in classes and modules. The compiler infrastructure is contained in the Runtime library as Ruby is a very dynamic language and new source code may be encountered or created at runtime.

2 Related Work

Jim Hugunin et al. have previously developed a .NET compiler for the Python language (2006). That effort was largely successful and reported speedups compared to the standard C-based Python implementation on Windows. The Python language is however, considerably simpler than Ruby.

MRI is widely acknowledged as one of the slower dynamic language implementations around. The official Ruby implementation will therefore soon change from MRI to an implementation based on a new virtual machine called YARV (Yet Another Ruby Virtual machine) developed by Koichi Sasada (2005).

Apart from those "standard" implementations, a number of alternative Ruby language implementations have emerged in recent years. One of the more notable is JRuby for the JVM platform. JRuby started out as an interpreter and has gradually morphed into a compiler.

On the .NET platform, John Lam has previously developed a Ruby/.NET bridge called RubyCLR that allowed MRI and .NET to inter-operate. This was not a true compiler, but a simple and practical means of integrating the two worlds.

Most recently, Microsoft has announced development of IronRuby for the .NET platform. The goals of IronRuby are very similar to those of Ruby.NET. IronRuby is a newer project and so is currently less developed than Ruby.NET. Parts of the Ruby.NET code base, including its scanner and parser have been incorporated into IronRuby.

While similar in goals, the approaches are slightly different. IronRuby is built on Microsoft's new Dynamic Language Runtime (DLR) - a .NET framework for implementing dynamic languages. The DLR is also used by IronPython and a new managed implementation of JScript. The DLR works by building an abstract syntax tree from Ruby source files and then dynamically converting them to Common Intermediate Language (CIL) code at runtime in a "Just in Time" fashion. The CIL code is then JIT compiled by the .NET JIT compiler. Ruby.NET, by comparison, statically compiles Ruby source files into .NET assemblies at compile time.

The DLR itself is simply a class library built on top of the standard .NET platform. It does not make use of any undocumented Microsoft internal system calls, so there is nothing that it can do that we are fundamentally unable to do also. IronRuby inherits both the advantages and disadvantages of using the DLR. It can leverage all of the optimization work that has gone into the DLR and it also allows IronRuby programs to easily interoperate with other DLR languages. However, the DLR is a large and complex framework, so even loading it takes some time, and at present the DLR is rather Python centric due to its design heritage and so may not be optimally tuned to Ruby's specific needs.

3 Challenges

3.1 Formal Specification

The first challenge that we faced was the lack of a formal specification for the Ruby language. This means that the most widely used implementation of Ruby – an interpreter implemented in C (RUBY, Thomas, Fowler & Hunt 2004) becomes the *de facto* specification. Fortunately, that implementation is open source and the license allows others to derive new works from it.

3.2 Parsing

Parsing modern programming languages is normally a relatively straight forward task. Unfortunately, Ruby's syntax is derived from Perl which is notoriously difficult to parse using traditional techniques. On first inspection the MRI grammar appears unnecessarily complex. For example, the syntax for array indexing is replicated in five separate contexts. MRI's YACC based parser is also tightly coupled to the hand written scanner with the two working closely together to resolve many potential ambiguities. We made extensive efforts to simplify the grammar and scanner but were ultimately defeated by the complexity and the lack of formal techniques for reasoning about the equivalence of different grammars.

Ultimately, we simply mirrored the implementation used in MRI. Even this was not a straight forward exercise as there were no reliable YACC like tools for the CLI. We were therefore forced to create our own tool – the Gardens Point Parser Generator (GPPG) (Kelly 2005). As we found, it was critical for our tool to behave virtually identically to YACC (or Bison). For example, when contemplating a reduce operation, the next input character should only be read if it is necessary to resolve between ambiguous reduce operations. This subtle behavior would go unnoticed in parsing most languages – but it was necessary for Ruby due to the tight interaction between the parser and scanner.

3.3 Compiling vs Interpreting

Ruby has traditionally been implemented as an interpreter. At "runtime" – the interpreter parses the Ruby source code, builds a tree data structure that represents the program, and then walks over this tree interpreting each node as it goes. Interpretation generally consists of two steps. The interpreter must first inspect the current tree node to *determine* what to do next, and then it must actually *perform* the appropriate operation.

A compiler does the parsing and determining what needs to be done at compile time. Compiled code therefore *generally* runs faster because at runtime because it only needs to *perform* the appropriate operations. Dynamic languages such as Ruby, however, introduce the possibility of new source code being constructed and encounter for the first time at runtime. In order to "compile" such languages we need the compiler infrastructure to be present at runtime so that we can dynamically compile and load the new code.

The dynamic semantics of languages such as Ruby also diminish the traditional performance advantages of compilation. Take for example the operation of adding two integers together. In a strongly typed language, the compiler will know at compile time that an integer addition is required and will be able to generate very efficient code. In Ruby, however, when we come across a `"+"` operator at compile time we cannot determine that it will be an integer addition as the *types* of the operands in the expression cannot be determined in general. Even if we did know the type of one of the operands, for example if the left operand was an integer literal, then we still couldn't be certain that an integer addition was required as someone may have overridden the `"+"` method for the `Fixnum` class. This overriding may have taken place in a separately compiled component, so we have no way of knowing whether this might have happened.

Now let us assume that the `"+"` method for the `Fixnum` class has not been overridden. The standard implementation for this method must still inspect the type of the right operand before we can determine the type of addition that is required. If the type of both operands is found to be integer then even then we can not simply add them together. We must check that the addition does not result in an overflow; otherwise we will need to promote the operands to `Bignum`.

As this example illustrates, the cost of the interpretation step tends to be relatively small compared to all the other tests that must be performed at runtime to achieve the correct dynamic semantics. The relative performance advantage of compilation is therefore diminished.

In the case of compiling for the CLI we face two further performance impediments compared to the native C-interpreter. Firstly, the compiled CIL code that we generate must be further just-in-time compiled at runtime into native code. Secondly our goal is to generate fully managed and verifiable CIL code. This means that the code we generate must be completely type safe, so we can not do the kind of pointer manipulation and type conversion tricks that the C-interpreter uses.

So, in summary, we believe it is possible to implement Ruby in a fully compiled manner, but we do not necessarily expect it to run significantly faster than MRI. The principal advantage then of producing a .NET compiler for Ruby is not improved performance, but providing the benefits of the CLI platform to Ruby programmers and to add Ruby to the set of languages that .NET programmers can choose from.

4 Mapping Ruby to the CLI

Ruby has objects, classes and methods. The CLI also supports objects, classes and methods so one might think compiling Ruby to the CLI to be a relatively straight forward exercise as it is with languages such as C#. However, as the previous section highlights, Ruby's dynamic semantics hinders such straight forward implementations.

Firstly, the good news - all Ruby objects belong to a class and the class that an object belongs to cannot change. The super-class from which a class inherits can also not change. The bad news is that the set of instance variables of an object and the set of methods belonging to a class can change at runtime. Variables and expressions are also not typed. So when we invoke a method, we generally do not statically know the type of the receiver object. Even in those cases where the type of the receiver can be inferred, we still don't statically know anything about the method that will bind to that method name at runtime – we don't even know how many parameters it will expect.

4.1 Ruby Classes

CLI classes have a fixed set of methods, so we cannot use CLI classes to represent Ruby classes. While we could dynamically generate CIL code for a Ruby class at runtime, once we have created a CLI class and created instances of it – it is then impossible to add or modify the set of methods that it supports. Ruby allows classes to be modified after instances of that class have been created. Such changes to a Ruby class can also occur in separately compiled source files, so it is generally impossible to have complete knowledge of a Ruby class at compile time. We therefore provide a CLI class called `Ruby.Class` to represent Ruby classes at runtime. These class objects contain a reference to their super-class (another `Ruby.Class` object) and a table that maps method names to the methods to which they are currently bound.

The process then of invoking a Ruby method is to:

1) Determine the Ruby class of the receiver object.
2) Determine the Ruby method currently bound to the specified method name for that Ruby class.
3) Invoke the found Ruby method.

4.2 Ruby Methods

Ruby methods need to be represented in such a way that they can be referenced in method tables. We could use CLI delegates for this purpose but since Ruby methods are not statically associated with a class, we choose to instead create a separate CLI class for each Ruby method. We use a singleton pattern to ensure only one instance of each Ruby method class is created – both for efficiency and for identity purposes.

These Ruby method classes contain a `Call` method which is used for invoking them. As these methods do not belong to the class of the receiver object, the receiver (or `self`) must be passed in as an explicit parameter. These compiler implementation details are of course completed hidden from application programmers.

If we had chosen to represent Ruby methods as delegates we would have had to choose a standard signature which would have involved passing an array of arguments. We avoid the overhead of allocating and initializing such an array in most common cases by providing overloaded `Call` methods catering for each number of arguments up to 10, plus a general purpose `Call` method for greater than 10 arguments. Each concrete Ruby method class overrides the `Call` method corresponding to the number of arguments that it expects. The abstract `RubyMethod` base class automatically takes care of calls to that method with other numbers of arguments.

4.3 Ruby Objects

All Ruby objects inherit from a Ruby class called `Object` which implements a number of standard methods such as `class`, `clone`, `freeze`, `inspect` and `methods`. An obvious approach therefore would be to require all objects used within Ruby programs to derive from a CLI class or interface called `Ruby.Object`. We do provide such a class but we do not assume that all objects passed as parameters to Ruby methods (or stored in Ruby collection classes) are derived from `Ruby.Object`. We instead allow any CLI reference type (derived from `System.Object`) to be used.

This has two advantages. Firstly, it supports interoperability with other .NET languages, without having to wrap other CLI objects within Ruby objects. Secondly, it allows us to represent primitive Ruby types more efficiently. For example, we can represent a Ruby

object of class *Fixnum* as a (boxed) *Int32*, Ruby objects of class *TrueClass* and *FalseClass* can be represented as boxed *Bool*s and Ruby objects of class *NilClass* are simply represented as a `null` reference.

So, if the only thing we know about an object is that it derives from *System.Object*, how can we usefully operate on it? What does a Ruby object need to be able to do?

4.4 Finding an Object's Ruby Class

To invoke a method on an object we first need to be able to determine its Ruby class. Rather than relying on the object itself to provide a field or method that returns its Ruby class, we instead provide a static method that takes a *System.Object* and returns its *Ruby.Class*. In most cases, the objects we use will derive from *Ruby.Object* – in which case the static method simply returns the *Ruby.Class* object stored in the *Ruby.Object*. If the object is *null*, a boxed *bool* or a boxed *int32* then the method return the ***Ruby.Class*** object corresponding to *NilClass*, *TrueClass*, *FalseClass* or *Fixnum* class, as appropriate. Otherwise the object must have been created by a component implemented in another .NET language. In that case, we dynamically create a special *Ruby.Class* object to represent the foreign CLI type. We maintain a static table that maps foreign CLI types to their corresponding *Ruby.Class* object, so that if we encounter an object of that type again, we can use the already created *Ruby.Class* object rather creating a new one. We use a special subclass of *Ruby.Class* called *Ruby.CLRClass* that uses CLI reflection to locate methods rather than the method table used by other *Ruby.Class* objects.

4.5 Instance Variables

In statically typed languages, the class defines the set of instance variable (or fields) that a class may possess, but the value of those instance variables is a property of each individual object. In Ruby, the class does not define a fixed set of instance variables. Each object of a particular class may have a different set of instance variables, and the set of instance variables associated with an object can change dynamically – separately compiled components may independently add to the set of instance variables that an object possesses. So, in general, the only way of representing instance variables is as a dictionary that maps instance variable names to their current value. And since instance variables are not typed, those values must simply be a reference to an object derived from *System.Object*.

It is therefore possible for different Ruby objects belonging to different Ruby classes to have the same CLI class (say *Ruby.Object*). These *Ruby.Object* objects will each contain references to their respective *Ruby.Class* objects and will each contain a field for their own instance variable dictionary. But what about *Fixnums*, *TrueClass*, etc? Their representations are not derived from *Ruby.Object*; where do their instance variables get stored? We use the same trick as MRI and store their instance variables in two dimensional look-aside table indexed by object and instance variable name.

4.6 Allocating Ruby Objects

As the previous section suggests, apart from a few special cases such as `int32`s and `bool`s, we can represent most Ruby objects using just the *Ruby.Object* CLI class. There are two reasons why we would not always want to do this. Firstly, the Ruby language defines a set of built-in Ruby classes such as *String*, *Array*, *Hash*, *Regexp*, *File*, *Dir* etc. Objects of these classes have an *intrinsic* value. For example a String encapsulates a list of characters. We therefore provide in our runtime library, separate CLI classes (derived from *Ruby.Object*) for each of the Ruby built-in classes. Each of these classes contains a private *value* field that is used by in our implementation of the methods for these classes. For example the *Ruby.String* class contains a *value* field of type *System.String*. We cannot directly represent Ruby strings as CLI strings as CLI strings are immutable whereas Ruby strings are mutable. The *Ruby.String* class contains methods to for example capitalize or reverse the string. These methods work by changing the `value` field of the *Ruby.String* object.

Obviously, then if a Ruby program calls the standard new method of the *String* class, we need to allocate a CLI object of class *Ruby.String* rather than just a *Ruby.Object*. If however, the Ruby program calls the new method of a class *Foo* that inherits from *String*, then we also need to ensure that we allocate a CLI object that at least derives `from` *Ruby.String*. The problem is that we cannot generally determine at compile time the super-class of a Ruby class. The super-class of a Ruby class is not allowed to change after the class has first been declared, but the super-class may be a runtime expression rather than being a statically known value. Even when the super-class is seemingly static, all may not be as it appears. Consider the following example:

```
class Foo < String

    ...

end
```

It might appear from this example that we know that class *Foo* inherits from the built-in *String* class. However, the super-class expression *String* is really just a "constant" expression, and in theory, the programmer could have previously redefined this constant to refer to some other *String* class – perhaps their own custom *String* class. This redefinition could have occurred in a separately compiled component, so in general, we have no way of knowing for certain what the base class of a Ruby class is at compile time.

When we are asked to create an instance of class *Foo*, we do not necessarily have to create an instance of a CLI

class called *Foo*, but if *Foo* does actually inherit from built-in class *String* then we need to ensure that we at least allocate an object that derives from *Ruby.String* (so that the implementation of the standard string methods can access its value field). Equally, if it turns out *String* is actually an alias for say built-in class *File*, then we need to at least ensure that we allocate an object that derives from class *Ruby.File*. The decision of which class to allocate must therefore be made at runtime. We active this by using Ruby's normal method binding mechanisms. We invoke a Ruby method called `allocate` to create the appropriate class of object. In this case, class *Foo* might not possess an `allocate` method of its own, but one of its superclasses will. If the parent-class turns out to be *String* (or derived from *String*), then the *String* `allocate` method will be encountered before the *Object* `allocate` method using Ruby's normal inheritance hierarchy search process.

The other reason we may not want to simply use an object of CLI class *Ruby.Object* to represent a Ruby *Foo* object is for interoperability purposes. Using a *Ruby.Object* object will achieve all of the appropriate semantics for a program implemented entirely in Ruby. But if we wish components implemented in other .NET languages to be able to conveniently use our Ruby classes, it is preferable if we create a CLI class specifically for Ruby class `Foo` and allocate such an object when required. But again, the problem is knowing statically what the base class is. So, in the above example, we would create a CLI class called *Foo* that inherits from *Ruby.String*, but at runtime we would have to decide whether the `allocator` for class *Foo*, ie the `allocator` that creates an instance of CLI class *Foo* was actually *safe* to use. If that `allocator` creates an object that does not derive from the type of object created by the base classes' `allocator` then we instead defer to using the base classes' `allocator`.

In either event, the CLI class *Foo* that we create is primarily a wrapper class that simply provides convenient access to that Ruby classes' methods and instance variables. We add CLI methods to this class corresponding to each of the statically known Ruby methods in the class, but these wrapper methods simply invoke the underlying Ruby methods via the normal dynamic Ruby method lookup process. In this way, the implementation of those methods can still change dynamically. The Ruby object may also gain additional dynamic methods that are not accessible via the static CLI wrapper methods but they can still be called using a more dynamic invoke API.

4.7 Local Variables

While the set of instance variables associated with an object is impossible to determine statically, the set of local variables used within a method or block is generally known at compile time. Each invocation of a method effectively gets its own copy of each of these variables. Such local variables would in less dynamic languages be allocated on the runtime stack as their lifetime corresponds to the duration of the corresponding method call. In Ruby, however, local variables may continue to live after the method that created them has returned. This can occur when a Ruby code block is created within that method. The code within the block has access to all of the local variables in its surrounding method. Such blocks can be treated as objects and be returned or otherwise escape the scope of the method. If this happens, the captured local variables need to remain live for as long as the block may be executed.

So, in general, rather than storing Ruby local variables as CLI local variables (which are stored on the runtime stack), we store Ruby local variables in special activation frame objects that we allocate on the CLI heap. At compile time we can generally determine which local variables will be used within each method, so we create a separate activation frame class (containing named fields for each local variable) for each Ruby method and block. The prologue of the `Call` method in each Ruby method class is responsible for allocating a new activation frame object. A reference to this frame object is stored in a CLI local variable so that it can be conveniently accessed in the remainder of the method. So, to access a local variable, we need to follow the CLI local variable to the activation frame and then access a named field within that activation frame. This is actually more efficient than the MRI implementation that traverses a list of local variables to find a match. If a nested block exists and escapes from the method then the block will contain a reference to the frame. The frame object will remain live and then naturally be garbage collected as soon as there are no longer any references to it.

4.8 Blocks

Ruby code blocks can also be nested inside other code blocks. Inner code blocks therefore have access to all local variables in surrounding blocks as well as the surrounding method. Blocks can be invoked just like Ruby methods so we represent each block by a class derived from *RubyMethod*. These custom created *Block* classes contain fields which point to each of its surrounding activation frame objects. When we access a local variable from within a nested block we always know statically at which nesting level it was defined. So to access such a local variable, we simply need to access the block's (strongly typed) field that corresponds to that level and then access the appropriately named local variable field within that activation frame.

4.9 Passing Parameters to Blocks

Blocks behave much like regular methods in that they both provide a list of formal arguments and they can be encapsulated in a *Proc* object and then be either called or invoked by `yield`. The processing of assigning actual parameters to formal parameters for blocks does however differ from the process used for methods. The assignment of actual parameters to formals for blocks is basically treated the same as Ruby's parallel assignment construct. So, for example, calling the block

```
{ |x, y, z| ...}
```

with arguments expr1, expr2, expr3 is equivalent to:

```
x, y, z = expr1, expr2, expr3
```

This however means that we can have "strange" formal argument lists such as:

```
{|a, (b, *c), c[a]| ... }
```

which contains duplicate arguments and L-value expressions.

Another important property of parallel assignment is that the syntax of the right hand side affects the semantics, not just its value. For example:

```
a, *b = [1,2,3]
```

produces different results to

```
a,*b = [1,2,3], *[]
```

even though their right hand sides have effectively the same value. A consequence of this is that if you pass these two different right hand sides to a method then the semantics will be the same, but if you pass them to a block via a `yield` command, then the semantics will be different (and if you pass them to a block via a `call` then the semantics are the same!)

This mean, when evaluating arguments lists, we need to maintain, not just the list of values computed, but also a flag indicating whether the syntax consisted of a single right hand side argument.

4.10 Dynamic Evaluation

Ruby provides a runtime method that takes a dynamically constructed string and interprets it as Ruby source code at runtime. In the simplest case, this is not difficult for us to achieve. We simply invoke our compiler at runtime and rather than writing the compiled code to a file, we write it to a memory stream, dynamically load the assembly and invoke the generated code. We use our own PE file writer for all code generation. Our writer is based on the published binary format specification used for CIL assembly files and avoids the verification steps performed by other CIL emitting APIs (verification is still performed by the CLI when the memory stream is loaded as an assembly).

The more complicated aspect of implementing the *eval* method is providing access to Ruby classes and local variables that already exist at runtime in outer contexts within which the *eval* method is invoked. Our implementation of the *eval* method first uses CLI reflection to determine the context within which it is invoked. It then sets up a static compiler context (abstract syntax tree) that encapsulates this runtime context. The given string is then parsed within this synthetically generated compile time context, so that all local variables encountered during parsing are automatically mapped to local variables and frame types that already exist at runtime.

The runtime context provided to the *eval* method might be the current runtime context or it may be a different runtime context as captured previously and encapsulated in a Ruby *Binding* object. A *Binding* object encapsulates the current `self` or receiver object as well as the current activation frame.

4.11 Dynamic Local Variables

Normally, as stated earlier, the set of local variables for a method or block can be statically determined. However, this is not true of local variables created by calls to the *eval* method. These dynamic local variables can, however, only be accessed by other calls to *eval* within the same frame. Other static code within the frame will not know of these local variables and will treat them as an undefined local or method. Consider for example:

```
def foo
    # x created dynamically within eval
    eval 'x = 42'

    # x is not treated as a local here
    puts x  # undefined local or method

    # but x is visible here inside an eval
    eval ' bar {|y| puts x }
end
```

Such dynamic local variables need to be accessed via a dictionary (similar to instance variables). But, as these dynamic local variables can only arise within *eval* code – they are relatively rare and we can lazily create a dictionary for them only if it turns out one is actually needed.

4.12 Non-local Control flow

One of the principal uses of code blocks in Ruby is as the body of a *for* or `each` loop. In this context, control flow constructs such as *break*, *retry*, *redo* and *return* make sense. *Break* leaves the block and continues after the loop, *redo* goes back to the start of the block, *retry* goes back to the start of the loop and *return* leaves the entire method. It must be remembered, however, that a block may escape from the method in which it is defined. If a control flow statement is executed in such a situation then the resulting control flow can be very non-local.

Consider the Ruby code below. A block is created within *methodA*. It is saved as a *Proc* object and later passed to *methodB* and subsequently passed as a block parameter to *methodC*. The "`yield`" in *methodC*

executes the block. The return statement causes control to return not just from the block or the method that invoked `yeild`, but all the way back to return from the method in which the block was declared (if it is still active).

```
def methodA()
    # create block
     x = proc { return 42; };
    # and then much later ...
    methodB(x);
    puts 'end methodA';
end

def methodB(y)
  methodC(&y);
  puts 'end methodB';
end

def methodC()
  yield();
  puts 'end methodC';
end

methodA();
```

In this case the block was declared within *methodA*, so control will leave the block, skip the end of *methodC*, skip the end of *methodB* and return from *methodA*. If the *return* statement is replaced by a *break* statement, then control will instead return to the end of *methodA*. A *retry* would cause *methodB* to re-execute (causing an infinite loop in this case).

Note: this non-local branching behavior only occurs if the block is invoked by `yield`. If we instead passed y as a *Proc* parameter to *methodC* and then called it, the *return* statement would only return control from the block back to *methodC*. The behavior also depends on how the block was defined, as *lambda* blocks behave differently to *proc* blocks and differently again if you pass a method as a *Proc*. All together there are about 24 semantic cases to consider – many of which are non-orthogonal and seem counter-intuitive. We assume many of these cases are simply consequences of the current MRI implementation rather than carefully considered language design choices.

In any event, our basic approach to achieving this kind of non-local branching behavior is to use CLI exceptions. We use three distinct exception classes for `return`, `retry` and `break`. If a *redo*, *break* or *retry* occurs within a loop then an appropriate branch instruction is generated. In contexts where non-local branching is required we instead throw an exception of the appropriate kind (*return*, *retry* or *break*). Each block has a reference to the frame of the scope in which it was defined. This defining scope is stored in the exception object when it is thrown, together with the any return value that may be provided by the control flow statement. We generate code that places every Ruby method call within its own `try` block which catches *Break* and *Retry* Exceptions. When one of these exceptions is caught, we check to see if the defining scope stored in the exception matches the current frame. If it does, then the appropriate frame has been found, so it branches to the appropriate label within that method and uses the return value stored in the exception. If the frames do not match, then the exception is re-thrown to the next outer call level. *Return* exceptions are caught by `try` blocks that we generate around each scope (class init, method or body) rather than the *try* blocks which surround each method call.

This process is potentially quite time consuming but it is only needed when such non-local branching constructs are encountered at runtime. In most cases control will return from a method call normally and none of the code in these catch blocks will ever be executed. This code pattern does however seriously increase the amount of code that we generate as it is replicated at each call site.

4.13 Continuations

Continuations allow a snapshot of the runtime stack to be made and to then return to that state at some latter point. We have not yet implemented continuations as there is still much discussion going on regarding whether they will be retained in future versions of the Ruby language. The CLI does not provide developers sufficient access to the CLI stack to perform these kinds of operations directly, so our intended approach for dealing with continuations also relies on CLI exceptions.

When a continuation is to be created, we would throw a special CLI continuation exception. The `try` block around each Ruby method call would also contain a special catch clause for this kind of exception. This catch block would save all of its local state (including the call point) into the continuation object and then re-throw the exception to the next call level for it to do the same. When this exception reaches the outermost level, we will have by that time captured all the state information needed for the continuation but unfortunately in the process completely pulled down the runtime stack. So, to be able to continue normally after having created the continuation, we then need to go about the process of restoring the CLI call stack to its original state.

Each method would have a special recreating parameter flag and the prologue code of each method would depending on this flag either execute the method normally, or jump to the call point where execution of this stack frame was previously. When a continuation is

called, the same process would be used to pull down the current call stack and to re-establish the stack of the continuation.

This approach relies on all of the methods on the CLI call stack being constructed according to a strict pattern to support the tearing down and building up of call stacks. This is fine for methods generated by our compiler. The hundreds of methods of Ruby's built-in classes and modules that we have implemented by hand would be more problematic. It does not work at all if there are methods on the stack from components implemented using other .NET languages.

4.14 Rescue Clauses

Ruby supports the raising of exceptions. These exceptions all inherit from a built-in Ruby class called `Exception` which we represent using a CLI class called `Ruby.Exception`. Since `Ruby.Exception` inherits from `Ruby.Object` and not `System.Exception`, it is necessary to use a separate CLI class called `RubyException` (which does inherit from `System.Exception`) to actually throw the exception. The `Ruby.Exeception raise` method generates a `RubyException` object and the two objects work together, with each containing a reference to the other.

Ruby also supports `rescue` clauses for catching Ruby exceptions. Each `rescue` clause contains a list of *exceptions* that it will catch. This exception list, however, is not necessarily a list of the names of the exception classes. In general, it is a list of expressions which are meant to evaluate to Ruby objects that support method `"==="`. This Ruby method (which may be programmer supplied) is used to determine whether the current exception matches the specified exception. So, rather than just catching a particular CLI exception based on the Ruby exception listed in the `rescue` clause, we must effectively catch all Ruby exceptions and then check to see if it was one we where meant to catch, and if not, re-throw it. For interoperability purposes, we also want to be able to catch arbitrary other CLI exceptions in Ruby rescue clauses by wrapping them in Ruby exceptions. Rather than trying to catch and convert CLI exceptions to Ruby exceptions as close to their origin as possible, we instead lazily only convert them when and if they are caught by a `rescue` clause. If a CLI exception is thrown that is not within a `rescue` clause then it remains as a pure CLI exception. We therefore need to catch *all* CLI exceptions at every rescue clause (not just those derived from `RubyException`). We must therefore be careful, to ensure that we do not trap CLI exceptions such as our `Break` exceptions that we use for the implementation of non-local control flow.

Ruby also supports a separate and slightly different exception mechanism based on `throw` and `catch` constructs, but that mechanism is implemented as part of the built-in class library rather than as a language feature. We use a separate CLI exception class to implement that type of exception.

4.15 Ruby Threads

The Ruby language defines its own threading model rather than relying on the threading model of the underlying platform. Ruby's thread model is often described as a "Green" thread model as it does not support the concurrent execution of its threads. The standard Ruby interpreter uses a single operating system thread and manually time slices between them at certain designated places within the runtime. Each thread is guaranteed at least 10ms of uninterrupted execution before possibly being switched out.

We have not yet implemented Ruby threads as there is still some debate regarding whether their semantics will change in future versions of the Ruby language. Our plan, however, for implementing them was to use separate CLI threads for each Ruby thread, but to carefully control the use of those threads to ensure that only one thread was executing at any given time and that switches between threads could only happen at designed places within the runtime.

We have, however, also tried to make our implementation as thread safe as possible, so that we can also support concurrent CLI threads. We have avoiding using global variables to represent quantities that should be specific to the currently executing thread. For example, we don't store the current class in a global variable as MRI does.

4.16 Code Generation Invariants

As explained in the previous sections, CLI exceptions are used as part of the Ruby.NET implementation to achieve non-local control flow. This means that all Ruby method calls need to be placed in their own `try-catch` block. One of the CLI's verification rules is that the CLI argument stack must be empty when entering a `try` block. This poses problems when method calls are nested. Consider for example:

```
expr1.Foo(expr2, expr3.Bar(expr4, expr5));
```

Normally this would be translated into:

```
Code for expr1

Code for expr2

{

Code for expr3

Code for expr4

Code for expr5

Call Bar

}

Call Foo
```

But if we put a `try` block around just the call to `Bar`, then the stack would contain *expr1* and *expr2* on entry to the block.

We therefore need to translate it into:

```
temp1 = Code for expr1;
temp2 = Code for expr2;
temp3 = Code for expr3;
temp4 = Code for expr4;
temp5 = Code for expr5;
try {
    load temp3
    load temp4
    load temp5
    temp6 = Call Bar
} catch …
try {
    load temp1
    load temp2
    load temp6
    Call Foo
} catch …
```

Obviously if any of the expressions are literal expressions or expressions that do not themselves involve method calls, then we can simply load that literal rather than storing it in a temp. Our code generation for method calls is then based around the idea of *pre-computing* each of the arguments. If the argument is a literal then the pre-computing step is a no-op, otherwise pre-computing computes the value and stores it in a temp. These temps are recycled once we are finished with them. This is a very important pattern as just about everything in Ruby is done via a method call. For example applying the '+' operator is a method call, determining whether a Ruby exception matches a particular Ruby `rescue` clause is a method call, etc.

4.17 Different Notions of Current Class

The Ruby language has at least three different notions of what the "current class" is in a given context.

One definition denoted *`ruby_cbase`* represents the current lexical class and is used for accessing constants and class variables. Another definition denoted *`ruby_class`* is used when defining and undefining methods and aliasing. This notion of current class may not be the same as the current lexical class if an *`eval`* method is in progress that is using a *`Binding`* from another context. The dynamic nature of this notion of current class requires us to propagate it via a parameter passed to all blocks. The third notion of current class, denoted *`last_class`* is used for making super calls. It is propagated via a parameter passed to every Ruby method and used by our *`FindSuperMethod`* method to ensure that when a super method is called, that we look for a method higher up the class hierarchy than where the currently executing method was defined. Without it, an infinite loop can result when invoking super methods.

4.18 Object Ids and Weak References

Ruby Objects support a method called *`id`* which maps to a unique integer associated with that object. MPI basically just returns the address of the object in question. It is not possible to do this on the CLI in a strongly typed, safe and verifiable manner. We don't however, need to return the address of the object, we can return any integer, provided it uniquely corresponds to this object. It is a simple process to simply generate the next integer id in sequence when a new object asks for its *`id`*. Once we have returned an *`id`* we need to store it with the object so that we can ensure we will return the same *`id`* if asked again.

The bigger challenge is implemented the reverse operation that locates a Ruby object given its *`id`*. This can be done by maintaining a reverse lookup table. The potential problem with this approach is that placing an object in this table will prevent it from being garbage collected. The solution is to use a special CLI class called *`System.WeakReference`* which maintains a reference to an object without preventing it from being garbage collected. We can query a *`WeakReference`* object at any time to determine if the object that it points to has been garbage collected. We use this same approach for implementing the *`each_object`* method of the build-in module *`ObjectSpaces`* which enumerates all of the currently live Ruby objects.

4.19 Dynamic Code Loading

We may need to dynamically generate CIL code in two circumstances, firstly for *`eval`* methods and secondly when another Ruby source file is loaded. If the source file in question has already been compiled into a dynamic link library and that library is newer than both the source file and the Ruby compiler itself, then we load the precompiled library and use it. Otherwise, we dynamically parse, generate code to a memory stream and then load the new assembly.

One of the issues with dynamically loaded assemblies is that references to them by other dynamically loaded assemblies are not automatically resolved in the way they are for assemblies that originate from disk. We therefore maintain a list of dynamically loaded assemblies and manually resolve to them by providing a custom handler for the *`AssemblyResolve`* event of our current application domain. One disadvantage of our dynamic generation and loading of assemblies is that they are never garbage collected (as occurs with the CLI light-weight code generation API used by IronPython (Hugunin 2006)).

4.20 Command Line Arguments

Ruby has its origins on UNIX systems which support a rich range of command line globbing functionality. By the time the command line arguments find their way to the Main method of a CLI executable some information has already been lost (for example, the name of the executable and various parameter quote characters). So, rather than relying on the `args` array provided to the `Main` function, we instead call:

```
System.Environment.GetCommandLineArgs()
```

and manually perform our own globbing (analogous to how the win32 version of MRI works).

4.21 Symbols and Interning

MRI "*interns*" all strings used as class names, method names, etc, in a string table and then subsequently represents them by their integer offset within this table. We have so far, not performed such an interning and represent all of our methods names etc as CLI strings. This makes comparison slower but avoids the interning step. It would be nice if we could perform the interning at compile time, but this is not possible if we are to support separate compilation of Ruby source files as two symbols in different files with the same name should always be treated as identical. Interning of method names if one of many optimizations that we will soon experiment with.

5 Optimization

Our focus so far has been exclusively on attaining the correct dynamic semantics of the Ruby language, and we have made no attempt to optimize the performance of the compiled code.

Nevertheless, we have attempted to avoid architectural features that would stand in the way of future efforts to improve the performance. Our aim is to incrementally add optimization steps in an environment where our regression test infrastructure can ensure that the changes do not affect correctness.

Interning method names as an optimization of dictionary lookup has already been mentioned in the previous section.

Mature implementations of dynamic languages invariably rely on call-site caching of method bindings to offset the overhead of repeated lookups. Although it is possible for such bindings to change, it is infrequent in practice. Thus caching is a big win. Such a mechanism is effective in the case of Ruby, and we expect to implement the mechanism shortly. The challenge is to find the least conservative rule for invalidating bindings that still ensures correct behaviour.

Finally, we are aware the computational data paths in the runtime libraries are far from optimal. As an example the main line of computation in the *equals* test for Ruby strings has a chain of method calls that between them perform no less than seven type instance tests before finally getting to start a character by character comparison. This can surely be improved, although it is a laborious task to ensure that all *non*-mainline control flow paths in the replacement code still provide correct semantics.

6 Conclusions

Our implementation of Ruby on the CLI has demonstrated that a verifiable code, compiled approach can achieve correct semantics. Our current implementation passes all tests for the functionality that we have implemented. As noted previously, the most significant missing functionality is threading and continuations. There is also some lack on functionality due to the fact that we have not implemented many of the libraries that are supplied as C-language interop libraries in the standard Ruby distribution. We expect that managed replacements for these will gradually be provided by us, or by others, as the need arises.

The "experiment" has highlighted once again the issues that arise in attempting new implementations of languages that are informally defined. In return we have been led rather more deeply into the unique detailed semantics of the Ruby language. It has been a challenging and rewarding journey, and we hope that our experience will be of assistance to others who choose to tread the same path.

7 Acknowledgements

We wish to thank Microsoft Research for their financial and technical support of this project.

8 References

GOUGH, J, Compiling for the .NET Common Language Runtime (CLI) Prentice Hall PTR, 2002

HUGUNIN, J. Jim Hugunin's Thinking Dynamic, http://blogs.msdn.com/hugunin/archive/2006/09/05/741605.aspx, Accessed 18 October 2007.

IRONRUBY, A fast, compliant Ruby powered by .NET, http://www.ironruby.net/, Accessed 18 October 2007.

JRUBY, Java powered Ruby Implementation, http://jruby.codehaus.org/, Accessed 18 October 2007.

KELLY, W. Gardens Point Parser Generator. http://www.plas.fit.qut.edu.au/gppg, Accessed 18 October 2007.

LAM J, RubyCLR, http://rubyforge.org/projects/rubyclr/, Accessed 18 October 2007.

MILLER, J & RAGSDALE, S. The Common Language Infrastructure Annotated Standard, Addison-Wesley Professional, 2003

RUBY. Ruby Language home page, http://www.ruby-lang.org, Accessed 18 October 2007.

SARADA K, YARV: Yet Another Ruby VM, http://www.atdot.net/yarv/, Accessed 18 October 2007.

THOMAS, D, FOWLER, C & HUNT, A. Programming Ruby, 2nd Edition, Pragmatic Bookshelf, 2004

Privacy Preserving Set Intersection Based on Bilinear Groups

Yingpeng Sang **Hong Shen**

School of Computer Science
The University of Adelaide,
Adelaide, South Australia 5005, Australia,
Email: {yingpeng.sang, hong.shen}@adelaide.edu.au

Abstract

We propose a more efficient privacy preserving set intersection protocol which improves the previously known result by a factor of $O(N)$ in both the computation and communication complexities (N is the number of parties in the protocol). Our protocol is obtained in the malicious model, in which we assume a probabilistic polynomial-time bounded adversary actively controls a fixed set of t ($t < N/2$) parties. We use a $(t+1, N)$-threshold version of the Boneh-Goh-Nissim (BGN) cryptosystem whose underlying group supports bilinear maps. The BGN cryptosystem is generally used in applications where the plaintext space should be small, because there is still a Discrete Logarithm (DL) problem after the decryption. In our protocol the plaintext space can be as large as bounded by the security parameter τ, and the intractability of DL problem is utilized to protect the private datasets. Based on the bilinear map, we also construct some efficient non-interactive proofs. The security of our protocol can be reduced to the common intractable problems including the random oracle, subgroup decision and discrete logarithm problems. The computation complexity of our protocol is $O(NS^2\tau^3)$ (S is the cardinality of each party's dataset), and the communication complexity is $O(NS^2\tau)$ bits. A similar work by Kissner et al. (2006) needs $O(N^2S^2\tau^3)$ computation complexity and $O(N^2S^2\tau)$ communication complexity for the same level of correctness as ours.

Keywords: cryptographic protocol, privacy preservation, bilinear groups, set intersection, non interactive zero-knowledge proof.

1 Introduction

For datasets distributed on different sources, computing the intersection without leaking the other elements is a frequently required task. One example is that one airline company is always required to find out those passengers who are on their private passenger list and the government's "do-not-fly" list. Another example is that some companies may decide whether to make a business alliance by the percentage of customers who have consumption records in all of them. In these scenarios, none of the companies or government is willing to to publish the other elements in their datasets than those of the intersection. In this paper, we address this problem as *Privacy Preserving Set Intersection* (PPSI), in which there are N ($N \geq 2$) parties, each party P_i ($i = 1, ..., N$) has a set (or multiset) T_i and $|T_i| = S$, each party wants to learn the intersection $TI = T_1 \cap ... \cap T_N$, without gleaning any information on the other parties' private elements except TI.

Generally speaking two types of *probabilistic polynomial-time* (PPT) bounded adversaries are considered in the research of *Secure Multiparty Computation* (SMC) : *semi-honest* (*passive*) and *malicious* (*active*). A semi-honest party is assumed to follow the protocol exactly as what is prescribed by the protocol, except that it analyzes the records of intermediate computations. A malicious party can arbitrarily deviate from the protocol. Theoretically if the adversary controls $N/2$ or more parties, a robust protocol can not be achieved to tolerate early-quitting of the malicious parties. More details on the semi-honest and malicious models can be found in other works (Yao 1982, Goldreich et al. 1987, Goldreich 2004). In this paper, we assume that the adversary corrupts a set of less than $N/2$ parties before the start, and maliciously controls the fixed set during the execution. The adversary we consider is also called *non-adaptive*. There are also *adaptive* adversaries that can select the parties they control as the execution proceeds, but they are not considered in this paper.

We also assume a physical broadcast channel exists for all parties where there is a public board whose content change can be publicly tracked. In practice this broadcast channel can be implemented by the Authenticated Byzantine Agreement in the point-to-point network. More details can be found in works of Lamport et al (1982) and Dolev et al (1983).

A PPSI protocol was proposed by Kissner et al. (2005) by constructing a randomized polynomial Y whose roots set contains the intersection set TI. In this paper, we construct a simplified Y, while still keeping the private relationship among Y and all T_i. Specifically we construct the encrypted Y by one cryptosystem which is semantically secure under the *Subgroup Decision Assumption* (SDA). Cryptosystems based on SDA generally require a limited plaintext space because after the decryption there is still one discrete log computation to recover the plaintext, as found in other works (Yamamura et al. 2001, Boneh et al. 2005). However, in this paper we show how they can be applicable on a plaintext space as large as the order of the subgroup on which the cryptosystems are constructed. A few candidate cryptosystems by Yamamura et al. (2001) can be considered for our protocol, but we choose the Boneh-Goh-Nissim (BGN) cryptosystem from Boneh et al. (2005), because it is based on the bilinear map by which we can construct efficient *non-interactive zero-knowledge* NIZK proofs.

Our PPSI protocol based on the NIZK proofs has $O(NS^2\tau^3)$ computation complexity, and $O(NS^2\tau)$

communication complexity. The PPSI protocol by Kissner et al. (2006) need $O(N^2S^2\tau^3)$ computation complexity and $O(N^2S^2\tau)$ communication complexity. Though they have stated that the communication complexity may be reduced to $O(N^2S\tau)$ by applying cut-and-choose technique, it has not been shown how this can be achieved and what is the reduced computation complexity.

The remainder of the paper is organized as following: Section 2 discusses some related work. Section 3 formally defines the problem of PPSI. Section 4 lists the basic tools for our protocol. Section 5 constructs the non-interactive proofs required in our protocol. Section 6 proposes the PPSI protocol for the malicious model. In Section 7 we analyze the computation and communication costs of our protocol. Section 8 concludes the whole paper.

2 Related Work

A solution for the multi-party case of PPSI was firstly proposed by Freedman et al. (2004). The solution is based on evaluating polynomials representing elements in the sets. Kissner et al. (2005) proposed a more efficient solution for PPSI, in which each polynomial representing each set is multiplied by a random polynomial which has the same degree with the former polynomial. The degree of the random polynomial was optimized by Sang et al. (2006). All these protocols are based on the semi-honest model.

Kissner et al. (2006) extended their PPSI protocol (Kissner et al. 2005) to the malicious model using zero knowledge proofs. The main idea of their protocol is constructing a polynomial $Y = \sum_{l=1}^{N}(f_l * \sum_{j=1}^{N} r_{i,l})$ where f_l is a polynomial having P_l's dataset T_l as the roots set, $r_{i,l}$ is a uniformly selected polynomial with the same degree with f_l. Specifically each P_i computes and broadcasts $y_i = f_1 * r_{i,1} + ... + f_N * r_{i,N}$, then they sums all y_i to get Y. With overwhelming probability, the roots set of Y equals TI. The major cost of their protocol is computing the N^2 polynomials multiplications $E(f_l * r_{i,l})$, given $E(f_l)$ and $r_{i,l}$, and then proving the correctness of each multiplication. Suppose $f_l = \sum_{k=0}^{S} a_{l,k}x^k$, $E(f_l) = \{E(a_{l,k})|k = 0, ..., S\}$, $r_{i,l} = \sum_{k=0}^{S} b_{i,l,k}x^k$, $E(\cdot)$ is an additive homomorphic encryption scheme, then each coefficient of $E(f_l * r_{l,k})$ can be computed by the corresponding coefficient-multiplications of $E(f_l)$ and $r_{i,l}$. It is easy to get that totally $(S+1)^2$ coefficient-multiplications are required, and each of them need one modular exponentiation. The proof of correct coefficient-multiplication is proposed by Cramer et al. (2001), which is constant size. Thus Kissner et al. (2006)'s protocol is $O(N^2S^2)$ size, i.e. the computation complexity is $O(N^2S^2\tau^3)$ (τ is the length of each element, $O(\tau^3)$ is the complexity of one modular exponentiation), and the communication complexity is $O(N^2S^2\tau)$ bits.

Kissner et al. (2006) also proposed that cut-and-choose technique may be used to simplify the proof of correct $E(f_l * r_{l,k})$, but cut-and-choose technique may compromise the correctness of the proof. For example, in a simplified proof the prover can firstly broadcast the $2S + 1$ coefficients of $E(f_l * r_{l,k})$, the verifiers randomly select a coefficient (by means of selecting random common challenge from Cramer et al. (2001)), then the prover proves the multiplications inside the selected coefficient are all correct. This proof is of $O(S)$ size, and reduces the communication complexity of the whole protocol to $O(N^2S\tau)$ bits, but will not reduce the computation complexity to $O(N^2S\tau^3)$. At least $O(NS^2\tau^3)$ computation cost is still required to compute N numbers of $E(f_l * r_{l,k})$ on each party. What's more, in this simplified proof the probability of one coefficient not being chosen by the verifiers is $1 - \frac{1}{2S+1}$. When S is large, this probability can not be negligible. Then a wrong coefficient will not likely be found out, by which a malicious party can multiply $r_{l,k}$ with another $E(f'_l)$ other than $E(f_l)$, then Y will not be correctly constructed, and its roots set will not equal TI. Some other techniques can be found to simplify the zero knowledge proof.

Sang et al. (2007) also extended their work (Sang et al. 2006) to get a PPSI protocol in the malicious model with $O(t^2S^2\tau^3)$ computation complexity, using interactive zero-knowledge proofs from Cramer et al. (2001), but there was still high cost polynomials multiplications in this extension. The complexity will be ideal if there is no polynomials multiplications in constructing Y, i.e., $Y = \sum_{l=1}^{N}(f_l * R_l)$ in which R_l is a random number. However, a malicious party knowing the coefficients of this Y will get advantage on guessing the coefficients of f_l from an honest party. In this paper we will construct this Y in its encrypted form, evaluate $E(Y)$ without leaking its coefficients.

Hohenberger et al. (2006) proposed a solution to two-part set disjointness testing, the security of which is based on the subgroup decision assumption, the same assumption with this paper. They also extended their solution to solve two-party set intersection problem, but the soundness will be violated in their extension, because a malicious party can reveal any element as an intersected element to the other party. To ensure the soundness, a commonly-shared polynomial Y can be constructed, and each party judges the intersection for himself by evaluating his elements in Y, as Kissner et al. (2006), Sang et al. (2007) and this paper have done.

3 Problem Definition

3.1 Assumptions and Major Notations

Suppose $\mathbb{T}$ is the domain of the inputs on all parties, we assume the size of $\mathbb{T}$, i.e. $|\mathbb{T}|$, is sufficiently large to prevent the dictionary attack. Specifically, we assume $|\mathbb{T}| \gg tS$. If tS is comparable with $|\mathbb{T}|$, the adversary controlling t parties may manipulate the t datasets to cover $\mathbb{T}$, then he can defraud the intersection of all honest parties' private inputs. We stress that this assumption is practical in the examples of airline company and business alliance, where the key of each record can be the customer's identity or passport number, and the domain of the key is much larger than tS.

We also assume the parties have negotiated an S that is not a sensitive privacy for each of them. The parties also do not mind the leakage of such information as all parties have less than S elements in their datasets. Then if a party has less than S elements, he can add dummy elements like $'|\mathbb{T}| + 1'$ to fulfill the size of S. The result of $'|\mathbb{T}| + 1'$ being in TI leaks no information except that all parties have less than S elements.

In Table 1, we define the major notations in this paper.

3.2 Definitions

In SMC, the security in both types (semi-honest and malicious) of adversaries is argued by the computational indistinguishability of the views in the ideal model and real model (as found in works of Goldreich et al. (1987), Lindell (2003)).

Table 1: Major Notations

Notation	*Definition*
N	Total number of parties
P_i	The i-th party
T_i	The set or multiset on P_i
S	Total number of elements on each party
$T_{i,j}$	The j-th element on P_i, $j = 1, ..., S$
$\overline{T}$	$(T_1, ..., T_N)$
TI	$T_1 \cap ... \cap T_N$
t	Total number of colluded parties, $t < N/2$
I	The index set of t colluded parties, $\{i_1, ..., i_t\}$
I'	The index set of honest parties, $\{i'_1, ..., i'_{N-t}\}$
τ	The security parameter which can be 256
$r \in_R \mathbb{Z}_n$	Uniformly select an element r from a group $\mathbb{Z}_n$

Definition 1 (**Computational indistinguishability**) *Suppose an ensemble $X = \{X_n\}$ be a sequence of random variables X_n for $n \in \{1, ..., M\}$, which are ranging over strings of length poly(n). Two ensembles $X = \{X_n\}$ and $Y = \{Y_n\}$ are **computationally indistinguishable**, denoted by $X \equiv^c Y$, if for every PPT algorithm A, and every $c > 0$, there exists an integer K such that for all $n \geq K$, $|Pr[A(X_n) = 1] - Pr[A(Y_n) = 1]| < \frac{1}{n^c}$. $Pr[A(x) = 1]$ is the probability that A outputs 1 on input x.*

Definition 2 (**Intersection Function** $\mathfrak{f}$) *The intersection function $\mathfrak{f}$ is an N-ary function: $(\{0,1\}^{\tau * S * N}) \rightarrow (\{0,1\})^{S*N}$, i.e., $\mathfrak{f}(\overline{T}) = \{\mathfrak{f}_{ij}(\overline{T}) | i = 1, ..., N, j = 1, ..., S\}$, where $\mathfrak{f}_{ij}(\overline{T}) = 1$ if $T_{i,j} \in TI$, and $\mathfrak{f}_{ij}(\overline{T}) = 0$ if $T_{i,j} \notin TI$.*

Definition 3 (**PPSI in the malicious model**) *Let Π be an N-party protocol for computing $\mathfrak{f}$. Let a pair (I, A), where A is a PPT algorithm representing an adversary in the real model, and $I = \{i_1, ..., i_t\}$ ($t < N/2$) is the index set of parties controlled by A. The joint execution of Π under (I, A) in the real model, denoted $REAL_{\Pi,I,A}(\overline{T})$, is defined as the output sequence of A and honest parties, resulting from their interaction in the execution of Π.*

Let a pair (I, B), where B is a PPT algorithm, represent an adversary in the ideal model, where there is an available trusted party. The joint execution of $\mathfrak{f}$ under (I, B) in the ideal model, denoted $IDEAL_{\mathfrak{f},I,B}(\overline{T})$, is defined as the output pair of B and the honest parties in the ideal execution.

*Π is said to **securely solve the problem of privacy preserving set intersection** in the malicious model, if for every PPT algorithm A, there exists a PPT algorithm B, such that the views of A and B are computationally indistinguishable, i.e.,*

$$\{IDEAL_{\mathfrak{f},I,B}(\overline{T})\} \equiv^c \{REAL_{\Pi,I,A}(\overline{T})\}. \quad (1)$$

4 Basic Tools

4.1 Bilinear Groups

Bilinear group $\mathbb{G}_1$ is a group which supports an *admissible bilinear map* $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ in which $\mathbb{G}_2$ have the same order with $\mathbb{G}_1$. A bilinear map is said admissible if it satisfies the following properties:

- *Bilinear:* $\forall u, v \in \mathbb{G}_1$, $\forall a, b \in \mathbb{Z}$, $e(u^a, v^b) = e(u, v)^{ab}$.
- *Non-degenerate:* If g is a generator of $\mathbb{G}_1$, then $e(g, g)$ is also a generator of $\mathbb{G}_2$.
- *Computable:* There is an efficient algorithm to compute $e(u, v)$ for any $u, v \in \mathbb{G}_1$.

In this paper we use the bilinear group $\mathbb{G}_1$ of composite order $n = q_1 q_2$ for two large primes q_1 and q_2, which is a subgroup of the additive group of points of an elliptic curve E over $\mathbb{F}_p$ ($p+1$ is divisible by n). An admissible bilinear map e for $\mathbb{G}_1$ can be the modified Weil pairing or Tate pairing over the curve (as found in works of Boneh et al. (2003), Miller (2004)). $\mathbb{G}_2$ is an order n subgroup of the multiplicative group of a finite field $\mathbb{F}^*_{p^2}$.

4.2 A Threshold Version of Boneh-Goh-Nissim Cryptosystem

We use the BGN cryptosystem (Boneh et al. 2005) based on the composite order bilinear group for its additive homomorphism and one-time multiplicative homomorphism. Boneh et al. (2005) has used its threshold version for electronic election, but the details on the key generation and decryption are not given. Below we give these algorithms.

- *Distributed Key Generation Algorithm* $\mathcal{G}$: Given a security parameter τ, a key generation algorithm $\mathcal{G}(\tau)$ can be run to get a tuple $(q_1, q_2, \mathbb{G}_1, \mathbb{G}_2, e)$. Specifically, q_1 and q_2 are two τ-bit random primes. Let $n = q_1 q_2$. Find the smallest positive integer $l \in \mathbb{Z}$ such that $p = ln - 1$ is prime and $p = 2 \mod 3$. In practice, τ can be 256 to get p of at least 512-bit length. The elliptic curve $y^2 = x^3 + 1$ defined over $\mathbb{F}_p$ has $p + 1 = ln$ points in $\mathbb{F}_p$. Then $\mathbb{G}_1$ is the subgroup of order n in the group of points on the curve. Let $\mathbb{G}_2$ be the subgroup of $\mathbb{F}^*_{p^2}$ of order n. The bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ is the modified Weil pairing on the curve. Select two random generators $g, u \in_R \mathbb{G}_1$ and set $h = u^{q_2}$. Then h is a random generator of the order q_1 subgroup of $\mathbb{G}_1$.

  The public key is $(n, \mathbb{G}_1, \mathbb{G}_2, e, g, h)$. The secret key $sk = q_1$. sk is distributed among the N parties as sk_i for $i = 1, ..., N$ by Shamir's secret sharing scheme (Shamir 1979). The verification key vk_i is also generated as h^{sk_i}. The public key and each share of secrete key can be generated by the technique of Frankel et al. (1998) and Pedersen (1991) without a trusted dealer. For simplicity we assume there is a trusted dealer that can run $\mathcal{G}$ as an offline phase.

- *Encryption Algorithm* E: To encrypt a message $m \in \{0, ..., 2^{\tau-1}\}$, select $r \in_R \mathbb{Z}_n$ and compute $C = E(m) = g^m h^r \in \mathbb{G}_1$.

- *Partial Decryption Algorithm* D_i: To decrypt a ciphertext C, party P_i computes and broadcasts $D_i(C) = C^{sk_i}$. Each party checks whether $e(C, vk_i) = e(D_i(C), h)$ to verify the correctness of $D_i(C)$. Because $e(C, vk_i) = e(g^m h^r, h^{sk_i}) = e(g, h)^{msk_i} e(h, h)^{rsk_i}$, and $e(D_i(C), h) = e(g^{msk_i} h^{rsk_i}, h) = e(g, h)^{msk_i} e(h, h)^{rsk_i}$, interactive zero knowledge proofs are not needed here.

- *Recovery Algorithm:* If less than $t + 1$ parties pass the verification of partial decryptions, the algorithm fails. Otherwise, suppose $\mathcal{S}$ be a set of $t + 1$ verified parties, and λ_i ($i \in \mathcal{S}$) be the appropriate Lagrange coefficients, the decryption $D(C) = \prod_{i \in \mathcal{S}} C^{\lambda_i sk_i} = C^{q_1} = g^{mq_1}$.

In our constructions we only need to know whether $m = 0$. For $m \in \{0, ..., 2^{\tau-1}\}$, $m = 0$ if and only if $D(C) = 1$. If $m \neq 0$, $D(C)$ is an element in the order q_2 subgroup of $\mathbb{G}_1$. To know m, a practical way is

computing firstly the discrete log mq_1, then the greatest common divisor $GCD(mq_1, n)$ to get q_1. Though the latter computation can be solved by Euclid's algorithm within polynomial time, the former problem is known as hard as the Discrete Log (DL) problem over a finite field (as proved by Menezes et al. (1993)). Our construction also utilizes the intractability of $m \neq 0$ in $D(C)$ to protect privacy.

The semantic security of BGN encryption is based on the hardness of *subgroup decision problem* as proved by Boneh et al. (2005): Given $(n, \mathbb{G}_1, \mathbb{G}_2, e)$ generated by $\mathcal{G}$ where $n = q_1q_2$, and an element $x \in \mathbb{G}_1$, output $'1'$ if the order of x is q_1 (i.e. x is over an order q_1 subgroup of $\mathbb{G}_1$), and output $'0'$ otherwise. We assume there is no PPT algorithm that can solve the subgroup decision problem efficiently.

The BGN encryption supports additive homomorphism. Given $C_1 = E(m_1)$ and $C_2 = E(m_2)$, it is easy to compute $E(m_1 + m_2) = C_1C_2$. It also supports one-time multiplicative homomorphism. Given $u = g^\alpha$, $h = g^{\alpha q_2}$ for a random $\alpha \in_R \mathbb{Z}_n$, $e(g,h) = e(h,g) = e(g,g)^{\alpha q_2}$, $e(h,h) = e(g,h)^{\alpha q_2}$, so $e(C_1, C_2) = e(g^{m_1}h^{r_1}, g^{m_2}h^{r_2}) = e(g,g)^{m_1m_2}e(g,h)^{m_1r_2+m_2r_1+\alpha q_2 r_1 r_2}$. $e(g,g)$ is of order n, $e(g,h)$ is of order q_1, $e(C_1, C_2)$ is an encryption of m_1m_2 over $\mathbb{F}^*_{p^2}$ and the decryption key is q_1.

4.3 Calculations on encrypted polynomials

In our protocol, we need do some calculations on encrypted polynomials. For a polynomial $f(x) = \sum_{i=0}^{m} a_i x^i$, we use $E(f(x))$ to denote the sequence of encrypted coefficients $\{E(a_i)|i = 0, ..., m\}$. Given $E(f(x))$, where E is an additive homomorphic encryption scheme, some computations can be made as following:

1) Evaluate $E(f(x))$ at a value v: $E(f(v)) = E(a_m v^m + a_{m-1}v^{m-1} + ... + a_0) = E(a_m)^{v^m} E(a_{m-1})^{v^{m-1}} \cdots E(a_0)$.
2) Given a constant scalar c, compute $E(c \cdot f(x)) = \{E(a_m)^c, ..., E(a_0)^c\}$.
3) Given $E(g(x))$ where $g(x) = \sum_{j=0}^{m} b_j x^j$, compute $E(f(x) + g(x)) = \{E(a_m)E(b_m), ..., E(a_0)E(b_0)\}$.

5 Non-interactive Proofs for the PPSI Protocol

5.1 Main Idea of the PPSI Protocol in the semi-honest model

Briefly, our protocol for PPSI is based on evaluating a randomized polynomial Y whose roots set contains the intersection.

1) Each P_i computes $f_i = (x - T_{i,1}) \cdots (x - T_{i,S}) = \sum_{k=0}^{S} c_{i,k}x^k$, and broadcasts $E(c_{i,k})$ for $k = 0, ..., S$.
2) For $i = 0, ..., N-1$,
   2.1) P_i selects $R_{i,l} \in_R \mathbb{Z}_n$, computes $E(R_{i,l} * f_l)$ for $l = i, ..., i + t \mod N$.
   2.2) P_i sums all $E(R_{i,l} * f_l)$ to get $E(y_i) = E(R_{i,i} * f_i + ... + R_{i,(i+t \mod N)} * f_{i+t \mod N})$, and broadcasts $E(y_i)$.
3) Each P_i sums all $E(y_i)$ to get $E(Y) = E(\ (R_{N-t,0} + ... + R_{N-1,0} + R_{0,0}) * f_0 + ... + (R_{N-t-1,N-1} + ... + R_{N-1,N-1}) * f_{N-1}\ ) = E(\sum_{l=0}^{N-1}(\sum_{i'=(l-t \mod N)}^{l} R_{i',l}) * f_l)$.
4) Each P_i evaluates $T_{i,j}$ in $E(Y)$ for $j = 1, ..., S$. P_i decrypts $E(Y(T_{i,j}))$ by the help of other t parties. If the decryption is 1, $T_{i,j} \in TI$; otherwise, it determines $T_{i,j} \notin TI$.

Since $\sum_{i'=(l-t \mod N)}^{l} R_{i',l}$ in Y is generated by $t+1$ parties, it is always a random number that can not be manipulated by the adversary.

One encryption scheme based on subgroup decision problem, such as the BGN encryption, is necessary for the security of the protocol. If we use some other encryption schemes in which $D(E(Y_{i,j})) = Y(T_{i,j})$, then one adversary controlling t ($t < N/2$) parties can compute the coefficients of Y by the Lagrange interpolation, with $S+1$ different evaluations he can get. From the coefficients of Y, the adversary will get some advantage on guessing the distribution of an honest party's inputs. However, in BGN encryption, the adversary can get only $|TI|$ evaluations, thus he can not know any of Y's coefficients.

What's more, a malicious (active) adversary controlling t parties may have attacks in the protocol. In step 1), after receiveing $E(f_i)$ from an honest party P_i, he can substitute his inputs with f_i by just broadcasting $E(f_i)$ to others. He can also broadcast zero polynomials, then the roots set of Y will be the intersection of all honest parties, and he can test this intersection in the evaluations. In step 2.2), he can broadcast an arbitrary $E(y_i')$, then the outputs of the protocol will also be arbitrary, because the roots set of Y will have no certain relationship with the original inputs. In step 4) he can evaluate some values on $E(f_i)$ from an honest P_i instead of on $E(Y)$, and ask for the decryptions of them, then test the inputs of P_i. We construct *non-interactive zero knowledge* (NIZK) proofs to prevent these attacks.

5.2 Proof of Correct Multiplication

Suppose the prover and verifier have the common input $a_1 = E(m) = g^m h^r$ where the prover does not know the plaintext m, the prover is required to prove he does a correct multiplication $a_2 = E(mR)$. This *Proof of Correct Multiplication* (POCM) can be denoted as $POCM\{R, s_1 | a_1 = g^m h^r, a_2 = g^{mR}h^{rR+s_1}\}$, which means the prover should prove knowing R, s_1 such that $a_2 = a_1^R h^{s_1}$. POCM can be based on the bilinear map e as following:

1) The prover generates $R, s_1, x, s_2 \in_R \mathbb{Z}_n$, computes $a_2 = a_1^R h^{s_1} = g^{mR}h^{rR+s_1}$, $a_3 = a_1^x h^{s_2} = g^{mx}h^{rx+s_2}$, $a_4 = H(a_1, a_2, a_3, pid, sid)$, $z_1 = x + a_4 R \mod n$, $z_2 = s_2 + a_4 s_1 \mod n$, then broadcasts a_2, a_3, z_1 and z_2. $H(\cdot)$ is a one-way hash function (e.g. SHA-2), *pid* is the unique identity of the prover, *sid* is the unique identity of the current session which POCM is belonging to.
2) The verifier computes $a' = H(a_1, a_2, a_3, pid, sid)$, outputs '1' if $e(a_1, g^{z_1})e(g, h^{z_2}) = e(a_2, g^{a'})e(a_3, g)$, and outputs '0' otherwise.

The correctness of POCM is easy to verify. $e(a_1, g^{z_1}) = e(g^m h^r, g^{x+a_4R}) = e(g,g)^{mx+ma_4R}e(g,h)^{rx+ra_4R}$, $e(g, h^{z_2}) = e(g,h)^{s_2+a_4s_1}$, $e(a_2, g^{a'}) = e(g,g)^{mRa'}e(g,h)^{rRa'+s_1a'}$, and $e(a_3, g) = e(g,g)^{mx}e(g,h)^{rx+s_2}$. Because $a' = a_4$, it is easy to see that $e(a_1, g^{z_1})e(g, h^{z_2}) = e(a_2, g^{a'})e(a_3, g)$.

The zero-knowledge property can be based on the subgroup decision problem of BGN cryptosystem and

the random oracle $H(\cdot)$. A simulator can randomly select m from a given domain, select $r, R, s_1, x, s_2 \in_R \mathbb{Z}_n$, and compute $a'_1 = g^m h^r$, $a'_2 = g^{mR} h^{rR+s_1}$, $a'_3 = g^{mx} h^{rx+s_2}$, $a'_4 = H(a'_1, a'_2, a'_3, pid, sid)$, $z'_1 = x + Ra'_4 \mod n$, $z'_2 = s_2 + s_1 a'_4 \mod n$. Because of the subgroup decision problem, distinguishing the distributions (a'_1, a'_2, a'_3) from (a_1, a_2, a_3) in the real execution of POCM is computationally hard. Then distinguishing a'_4 from a_4 is computationally hard in a random oracle $H(\cdot)$, and it is also hard to distinguish (z'_1, z'_2) from (z_1, z_2) because x, R, s_1, s_2 in them are all uniformly selected. Therefore, we can say that the views of the simulator and the adversary-controlled verifier in POCM are computationally indistinguishable.

In POCM the prover computes 4 *exps* (i.e. modular exponentiations in $\mathbb{G}_2$ or point multiplications in $\mathbb{G}_1$), 4 *multis* (i.e. modular multiplications in $\mathbb{G}_2$ or point addition in $\mathbb{G}_1$), and one hash value. The verifier computes 3 exps, 2 multis, one hash value and 3 pairings. 4 messages need to be broadcasted.

5.3 Proof of Knowing Plaintext

The *Proof of Knowing Plaintext* (POKP) can be denoted as $POKP\{m, r | a_1 = g^m h^r\}$, which means the prover should prove that he knows m, r such that $a_1 = E(m) = g^m h^r$. POKP can be based on the bilinear map e as following:

1) The prover generates $x, s \in_R \mathbb{Z}_n$, computes $a_2 = g^x h^s$, $a_3 = H(a_1, a_2, pid, sid)$, $z_1 = x + a_3 m \mod n$, $z_2 = s + a_3 r \mod n$, then broadcasts a_2, z_1, z_2.
2) The verifier computes $a' = H(a_1, a_2, pid, sid)$, checks whether $e(g^{z_1} h^{z_2}, g) = e(a_1, g^{a'}) e(a_2, g)$, outputs '1' if it is the case, and outputs '0' otherwise.

POKP can be treated as a special case of POCM, i.e. given a common input $E(1)$, the prover should prove that he knows m such that $a_1 = E(1 \cdot m)$. Thus the correctness and zero-knowledge property of POKP are also easy to get. Including the computation of a_1 the prover computes 4 exps, 4 multis, one hash value, broadcasts 3 messages. The verifier computes one hash value, 3 exps, 2 multis and 3 pairings.

5.4 Proof of Correct Polynomial Evaluation

Suppose $E(Y) = \{E(c_i) | i = 0, ..., S\}$ for polynomial $Y = \sum_{i=0}^{S} c_i x^i$ is the common input, the prover is required to prove that he correctly evaluates a value v on $E(Y)$. This *Proof of Correct Polynomial Evaluation* (POCPE) can be denoted as $POCPE\{v | \forall i = 0, ..., S, E(c_i) = g^{c_i} h^{\gamma_i}, E(Y(v)) = E(\sum_{i=0}^{S} c_i v^i)\}$. POCPE can be constructed based on POKP, and POCM:

1) The prover proves knowing the plaintext of $E(v)$ by $POKP\{v, r_1 | a_1 = g^v h^{r_1}\}$.
2) For $i = 2, ..., S$, given $a_1 = g^v h^{r_1}$ and $a_{i-1} = g^{v^{i-1}} h^{r_{i-1}}$, the prover proves knowing some v and $r_i \in_R \mathbb{Z}_n$ such that $a_i = g^{v^i} h^{r_i}$. This proof is a simplified POCM as following:
   2.1) The prover generates $r_i \in_R \mathbb{Z}_n$, computes $a_i = g^{v^i} h^{r_i}$, $b_i = g^{-r_i + v r_{i-1}} a_{i-1}^{r_1} = g^{-r_i + v r_{i-1} + v^{i-1} r_1} h^{r_1 r_{i-1}}$, and broadcasts a_i, b_i.
   2.2) The verifier checks whether $e(a_1, a_{i-1}) = e(a_i, g) e(b_i, h)$.
3) Then the prover proves he correctly computes $E(Y(v)) = E(\sum_{i=0}^{S} c_i v^i)$:
   3.1) The prover generates $R \in_R \mathbb{Z}_n$, computes $c = E(Y(v)) = h^R \prod_{i=0}^{S} (E(c_i))^{v^i} = g^{\sum_{i=0}^{S} c_i v^i} h^{\sum_{i=0}^{S} \gamma_i v^i + R}$, $d = g^{-R} \prod_{i=1}^{S} (E(c_i))^{r_i} = g^{-R + \sum_{i=1}^{S} c_i r_i} h^{\sum_{i=1}^{S} \gamma_i r_i}$, and broadcasts c, d.
   3.2) The verifier checks whether $e(c, g) e(d, h) = e(E(c_0), g) \prod_{i=1}^{S} e(E(c_i), a_i)$.

The correctness and zero-knowledge property of POCPE are also easy to get since POCPE is composed of POKP, POCM. In sum the prover computes $6S + 2$ exps, $4S + 3$ multis and one hash value, broadcasts $2S + 3$ messages. The verifier computes 3 exps, $2S + 2$ multis, one hash value and $2S + 3$ pairings.

6 The PPSI Protocol in the Malicious Model

We add zero knowledge proofs into the PPSI protocol for the semi-honest model to get a protocol in the malicious model. A malicious adversary will be forced to behave in the semi-honest manners, otherwise he will be found cheating by these proofs. In Figure 1, we give the PPSI protocol for the malicious model. As analyzed in Section 5.1, the malicious behaviors should be prevented as following:

1) To prevent the adversary from generating zero polynomials, notice that $c_{i,S}$, the leading coefficient of f_i, should always be 1, so in step 2.2) of Figure 1, each party will set the leading coefficient $E(c_{i,S}) = E(1)$ by themselves, then $E(f_i)$ from P_i will have at least one nonzero coefficient, the roots set of Y will not equal the intersection of the honest parties.
2) To prevent the adversary from replacing his inputs by an honest party, ideally each party should run S times of POKP on its encrypted coefficients in step 1) of Figure 1. However, one POKP for any encrypted coefficient other than the leading one (say, POKP for $E(c_{i,S-1})$) suffices to prove the party generates its polynomial independently. The adversary may substitute his other coefficients (say, $E(c_{i,S-2})$, ..., $E(c_{i,0})$) with the coefficients received from an honest party, but any of his substitution will generate a polynomial whose roots are not known by him.
3) To prevent the adversary from broadcasting arbitrary encryptions in step 2.1) of Figure 1, each party should run POCM to prove the multiplications are correct.
4) To prevent the adversary from evaluating $E(f_i)$ from an honest P_i in step 4.1) of Figure 1, each party should prove the evaluations are correct by POCPE.

In Figure 1, if there is some party removed from the verified list in Step 1), only the intersection of the remaining parties' datasets will be computed. For the party removed in Step 2), any remaining party can be elected to act his part in the protocol. In Step 3), a removed party will not know the intersection. At least there are $t+1$ parties left in the final decryption, from which they know the final intersection. The protocol can also be applicable when $N \geq 2$ and $t \geq N/2$, but if there are less than $t+1$ parties remained in the final decryption, they can not get the final intersection.

Inputs: There are N parties, t of them may collude in malicious manners. Each party has a private set of S elements, denoted T_i. Each party holds the public key and its own share of the secret key for the $(t+1, N)$-threshold BGN cryptosystem. All party have a common session identification number *sid* for running the current protocol in the computing system. Each party has a unique identity *pid*.

Output: Each party P_i knows $TI = T_0 \cap ... \cap T_{N-1}$.

Steps:

1) Computing $E(f_i)$: For $i = 0, ..., N-1$, P_i computes $f_i = (x - T_{i,1}) \cdots (x - T_{i,S}) = \sum_{k=0}^{S} c_{i,k} x^k$, encrypts $c_{i,k}$ for $k = 0, ..., S-1$, broadcasts all the encrypted values, and then runs $POKP\{c_{i,S-1} : E(c_{i,S-1})\}$ to prove he knows the plaintext of $E(c_{i,S-1})$.

2) Computing $E(y_i)$: For $i = 0, ..., N-1$,

2.1) P_i selects $R_{i,l} \in_R \mathbb{Z}_n$, computes $E(R_{i,l} * f_l)$ for $l = i, ..., i+t \mod N$, and then broadcasts the coefficient $E(R_{i,l}c_{l,k})$ for $k = 0, ..., S-1$, runs $POCM\{R_{i,l} : E(R_{i,l}c_{l,k})\}$ to prove he does the correct multiplication.

2.2) each party sets $c_{l,S} = 1$, $E(R_{i,l}c_{l,S}) = E(R_{i,l})$, sums $E(R_{i,l} * f_l)$ for $l = i, ..., i+t \mod N$, to get $E(y_i) = E(R_{i,i} * f_i + ... + R_{i,(i+t \mod N)} * f_{i+t \mod N})$.

3) Each P_i sums all $E(y_i)$ to get $E(Y) = E(\ (R_{N-t,0} + ... + R_{N-1,0} + R_{0,0}) * f_0 + ... + (R_{N-t-1,N-1} + ... + R_{N-1,N-1}) * f_{N-1}\ ) = E(\sum_{k=0}^{S} \beta_k x^k)$.

4) Decryption and Evaluation: For $i = 0, ..., N-1$,

4.1) For $j = 1, ..., S$, P_i evaluates $T_{i,j}$ in $E(Y)$ to get $E(\beta_k T_{i,j}^k)$ for $k = 1, ..., S$, then runs $POCPE\{T_{i,j} : E(\beta_k T_{i,j}^k), k = 1, ..., S\}$ to prove the correctness of the evaluation.

4.2) For $j = 1, ..., S$, all party of the $t+1$ quorum including P_i compute $E(Y(T_{i,j})) = \prod_{k=0}^{S} E(\beta_k T_{i,j}^k)$. P_i decrypts $E(Y(T_{i,j}))$ by the help of the other t parties. If the decryption is 1, $T_{i,j} \in TI$; otherwise, it determines $T_{i,j} \notin TI$.

Figure 1: Protocol for Privacy Preserving Set Intersection in the Malicious Model

We show below briefly the correctness and security of our protocol, though these claims can also be mathematically proving.

Correctness If $T_{i,j} \in TI$, then $f_l(T_{i,j}) = 0$ for $l = 0, ..., N-1$. Then the evaluation $Y(T_{i,j}) = 0$ and $D(E(\ Y(T_{i,j})\ )) = g^{0q_1} = 1$. If $T_{i,j} \notin TI$, then there exists $f_{l'}(T_{i,j}) \neq 0$ for some $l' \in \{0, ..., N-1\}$. Then $Y(T_{i,j}) = \sum_{l'} (\sum_{i'=(l'-t \mod N)}^{l'} R_{i',l'}) * f_{l'}(T_{i,j})$, $D(E(\ Y(T_{i,j})\ )) = g^{q_1 Y(T_{i,j})}$, whose order is q_2. Because each $R_{i',l'} \in_R \mathbb{Z}_n$, overwhelmingly $Y(T_{i,j}) \neq 0$ and $D(E(\ Y(T_{i,j})\ )) \neq 1$.

Security Suppose A is the adversary in the real execution of PPSI protocol in Figure 1 which controls parties $P_I = \{P_{i_1}, ..., P_{i_t} | t < N/2\}$, B is the adversary which controls the same parties in the ideal execution assuming there is a trusted party $\mathcal{T}$. Based on the zero-knowledge properties of POCM, POKP and POCPE, B's views in the simulations of these proofs are computationally indistinguishable from A's views in the real execution of them. The zero-knowledge properties of our NIZK proofs are based on the basic assumptions of random oracle model, the subgroup decision and discrete logarithm problems, so by the same assumptions, the views of A and B are computationally indistinguishable from each other, which accounts for the security of our PPSI protocol as defined in Definition 3.

7 Complexity of the PPSI Protocol

The computation complexity of our protocol is mainly due to the encryptions, exponentiations, bilinear maps and multiplications. One BGN encryption needs 2 exponentiations. One exponentiation in $\mathbb{G}_2$ with a τ-bit exponent (or point multiplication in $\mathbb{G}_1$) requires $O(\tau^3)$ computation. One bilinear map, such as Weil pairing by Miller's algorithm, has the same complexity with the exponentiation in $\mathbb{G}_2$. One multiplication of two τ-bit elements in $\mathbb{G}_2$ (or point addition in $\mathbb{G}_1$) has cost of $O(\tau^2)$. One element in $\mathbb{G}_1$ has $O(\tau)$ bits.

Computation Complexity : We assume the N parties execute the protocol in parallel, and the computation cost of one party P_i can be representative of the whole protocol's complexity. In Step 1), P_i computes $E(f_i)$ by $2S$ exps, computes $POKP$ by 2 exps, 3 multis and one hash value, checks $POKP$ by $3(N-1)$ exps, $3(N-1)$ bilinear maps $2(N-1)$ multis and $N-1$ hash values. In Step 2.1), P_i computes all $POCM$ by $4(S+1)(t+1)$ exps, $4(S+1)(t+1)$ multis and $(S+1)(t+1)$ hash values, checks $POCM$ by

$3S(t+1)(N-1)$ exps, $3S(t+1)(N-1)$ bilinear maps, $2S(t+1)(N-1)$ multis and $S(t+1)(N-1)$ hash values. In Step 2.2), P_i computes all y_i by $(S+1)tN$ multiplications. In Step 3), P_i computes Y by $S(N-1)$ multiplications. In Step 4.1), for each evaluation of $T_{i,j}$, P_i computes one $POCE$, thus for S evluations P_i computes $6S^2+2S$ exps, $4S^2+3S$ multis and S hash values. P_i also checks $S(N-1)$ $POCE$ for other parties, which need $(2S+3)S(N-1)$ bilinear maps, $3S(N-1)$ exps, $S(N-1)$ hash values and $(2S+3)S(N-1)$ multis. In Step 4.2), P_i computes S decryptions, which need S exps, $2tS$ bilinear maps, and tS multis. P_i also computes $(N-1)S$ exps for other parties' decryptions.

The complexity for one exp (or bilinear map) is $O(\tau)$ times of that of one multiplication or hash value, so the complexity for all exps plus all bilinear maps can be representative for the whole protocol's complexity, as we have done in calculating Kissner et al. (2006)'s computation complexity. We also consider only the practical instance that $S \gg t$, i.e. $O(StN + NS^2) = O(NS^2)$. The major cost of our protocol is in Step 4.1), for the computation of $O(NS^2)$ bilinear maps. The whole protocol's computation complexity is $O(NS^2\tau^3)$.

Communication Complexity : In Step 1), $N(S+3)$ elements are broadcasted by the N parties. In Step 2.1), $4N(S+1)(t+1)$ elements are broadcasted. In Step 4.1), $NS(2S+3)$ elements are broadcasted. In Step 4.2), NtS elements are broadcasted. The major cost is in Step 4.1), the communication complexity is $O(NS^2)$ elements, or $O(NS^2\tau)$ bits.

8 Concluding Remarks

We proposed a more efficient privacy preserving set intersection protocol which improves a previous work by Kissner et al. (2006) by an $O(N)$ factor in the computation and communication complexities, with the same level of correctness. Though cut-and-choose technique may reduce the communication complexity of Kissner et al. (2006)'s protocol, the correctness may be compromised and the reduced computation complexity is the same with us. We proved the security of our protocol in the malicious model assuming an adversary actively controlling a fixed set of t ($t < N/2$) parties. Our construction is based on the BGN cryptosystem which supports bilinear maps. Efficient NIZK proofs for correct multiplications, knowing the plaintext, and correct polynomial evaluation are also constructed. In the future we will utilize the bilinear group and NIZK proofs in other privacy preserving set operations.

References

Boneh, D. & Franklin, M. (2003), Identity-Based Encryption from the Weil Pairing, *in* 'SIAM Journal of Computing', Vol. 32(3), pp. 586–615.

Boneh, D., Goh, E. & Nissim, K. (2005), Evaluating 2-DNF Formulas on Ciphertexts, *in* 'Proc. of TCC'05', Vol. 3378, LNCS, pp. 325–341.

Barreto, P., Kim, H., Lynn, B., & Scott, M., (2002), Efficient Algorithms for Pairing-Based Cryptosystems, *in* 'Proc. of CRYPTO'02', Vol. 2442, LNCS, pp. 354–369.

Cramer, R., Damgard, I. & Nielsen, J. (2001), Multiparty Computation from Threshold Homomorphic Encryption, *in* 'Advances in Cryptology - EUROCRYPT 2001', Vol. 2045, LNCS, pp. 280–300.

Dolev, D. & Strong, H. (1983), Authenticated Algorithms for Byzantine Agreement, *in* 'SIAM J. Comput', Vol. 12(4), pp. 656–665.

Frankel, Y., MacKenzie, P. & Yung, M. (1998), Robust efficient distributed RSA-key generation, *in* 'Proc. of the 17th annual ACM symposium on Principles of distributed computing', ACM Press, pp. 320–330.

Freedman, M., Nissim, K. & Pinkas, B. (2004), Efficient Private Matching and Set Intersection, *in* 'Proc. of Eurocrypt'04', Vol. 3027, LNCS, pp. 1–19.

Goldreich, O. (2004), Foundations of Cryptography: Volume 2, Basic Applications, Cambridge University Press, 2004.

Goldreich, O., Micali, S. & Wigderson, A. (1987), How to Play Any Mental Game, *in* 'Proc. of 19th STOC', ACM Press, pp. 218-229.

Hohenberger, S. & Weis, S. (2006), Honest-Verifier Private Disjointness Testing Without Random Oracles, *in* '6th International Workshop of Privacy Enhancing Technologies (PET 2006)', Vol. 4285, LNCS, pp. 277–294.

Kissner, L. & Song, D. (2005), Privacy-Preserving Set Operations, *in* 'Advances in Cryptology - CRYPTO 2005', Vol. 3621, LNCS, pp. 241–257.

Kissner, L. & Song, D. (2006), Privacy-Preserving Set Operations, *in* 'Technical Report CMU-CS-05-113', Carnegie Mellon University, June 2006.

Lamport, L., Shostack, R. & Pease, M. (1982), The Byzantine Generals Problem, *in* 'ACM Trans. on Programming Languages and Systems', Vol. 4(3), ACM Press, pp. 382–401.

Lindell, Y. (2003), Parallel Coin-Tossing and Constant-Round Secure Two-Party Computation, *in* 'Journal of Cryptology', Vol. 16(3), pp. 143-184.

Menezes, A., Vanstone, S., & Okamoto, T. (1993), Reducing elliptic curve logarithms to logarithms in a finite field, *in* 'IEEE Trans. on Information Theory', Vol. 39, pp. 1639–1646.

Miller, V. (2004), The Weil Pairing, and Its Efficient Calculation, *in* 'Journal of Cryptology', Vol. 17, pp. 235–261.

Pedersen, T. (1991), A Threshold Cryptosystem without a Trusted Party, *in* 'Proc. of Eurocrypt 1991', Vol. 547, LNCS, pp. 522–526.

Sang, Y., Shen, H., Tan, Y. & Xiong, N. (2006), Efficient Protocols for Privacy Preserving Matching Against Distributed Datasets, *in* 'Proc. of the 8th International Conference on Information and Communications Security (ICICS '06)', Vol. 4307, LNCS, pp. 210–227.

Sang, Y. & Shen, H. (2007), Privacy Preserving Set Intersection Protocol Secure Against Malicious Behaviours, *accepted by* 'The 8th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2007)', Adelaide, Australia, Dec. 2007.

Shamir, A. (1979), How to Share a Secret, *in* 'Communications of the ACM', Vol. 22(11), ACM Press, pp. 612–613.

Yamamura, A. & Saito, T. (2001), Private Information Retrieval Based on the Subgroup Membership Problem, *in* 'Australian Conference on Information Security and Privacy (ACISP'01)', Vol. 2119, LNCS, pp. 206–220.

Yao, A. (1982), Protocols for Secure Computations, *in* 'Proc. of the 23rd Annual IEEE Symposium on Foundations of Computer Science', pp. 160–164.

A Local Broker Enabled MobiPass Architecture for Enhancing Trusted Interaction Efficiency

Will Tao, Robert Steele
Department of Computer Systems, Faculty of Information Technology
University of Technology, Sydney
PO Box 123, Broadway, 2007, New South Wales
{wtao, rsteele}@it.uts.edu.au

Abstract

While mobile computing provides a potentially vast business opportunity for many industry participants, it also raises issues such as security and performance. This paper proposes a Local Broker enabled MobiPass architecture based on our previous research outcomes. Our MobiPass architecture can convert the unpredictable and highly dynamic mobile environment into a trusted business platform. By setting customised rules against a MobiPolicy, the Mobipass architecture enables fine grained access control without necessarily having a prior knowledge or interaction with other encountered parties and environments. This paper extends our MobiPass architecture by introducing an additional element – the Local Broker, to enhance the architecture's performance and efficiency. A detailed case study has been provided to explain the role that the Local Broker takes in the architecture.

Keywords: Mobile Computing, Ubiquitous Computing, Trusted Interaction

1 Introduction

Recent advances in technology have provided portable devices such as the mobile phone, personal digital assistant (PDA), portable data terminal (PDT) and smart phone with wireless computing capabilities. This kind of wireless computing model is often referred to by the generic term "mobile computing" and has already attained a substantial fundamental role in the business world.

However, to gain wide acceptance and success with this computing model, certain conditions will need to be satisfied before applying mobile computing into a critical, enterprise level system. An example of an inhibitor that deters mobile computing is that it is very difficult to build a trusted environment among all transacting entities within a mobile environment as it is highly dynamic and unpredictable (Satyanarayanan 2000, Ranganathan 2004). Unlike the traditional computing environment that is static and closed, with fixed, well-known entities within the network, mobile computing involves a large number of interactions, co-ordinations and collaborations with a large number of casually accessible yet portable mobile devices. The strategy and approach in building a trusted environment is fundamentally difficult and different when compared with more static networks.

In the case where there is a limited amount of knowledge about different transacting entities, a feasible mechanism that protects sensitive information and determines the level of trust between those entities in the mobile computing network is essential, as a lack of trust can result in failure to implement business models that build on top of this mobile environment. In addition, users will not be willing to participate as they do not have confidence in interacting with each other.

In this paper, based on our previously proposed MobiPass architecture, we put forward an alternative approach to establish a trusted interaction in mobile computing. The new approach introduces the new element, Local Broker (LB) into the architecture that will enhance performance, flexibility and other aspects. The case study, described in Section 4, will clearly illustrate the architecture.

The paper is structured as follows – Section 2 provides a review of the MobiPass architecture with a brief explanation, and Section 3 describes the Local Broker based MobiPass architecture. A case study is examined in Section 4 and in Section 5 related work is discussed followed by future work and the conclusion in Section 6.

2 MobiPass Architecture Review

The purpose of the MobiPass architecture is to help mobile entities to establish trusted interactions and provide a fine grained information access control among those transacting entities. The definition of trust in this paper is defined as "a subjective expectation about other's future behavior" as we believe that the trusted platform is a necessary and non-replaceable condition that can enable mobile computing to achieve a higher level of success.

MobiPass is a generic architecture that creates a flexible and secure environment in mobile computing; it can be applied to a large number of scenarios where trusted interaction is required. As the MobiPass architecture utilises and extends digital certificate technologies to provide more detailed certified information in a *distributed* manner, it is not necessary to have a central server to implement trusted interaction among large numbers of mobile entities. This distributed nature is a critical attribute, given the vast number of potential mobile entity interactions for which trusted interaction must be achieved. To clearly describe the MobiPass architecture, we will use the mobile phone as an example to

demonstrate how this architecture works within a mobile computing environment.

By enabling a set of customised preset preferences, the MobiPass architecture allows mobile entities which are previously unknown to each other to interact and communicate in a trusted manner. In the architecture, the mobile entity only talks with and makes itself visible to the trusted entity/environment which satisfies the customised access control rules.

The core elements in the MobiPass architecture are: The Central Registry, MobiPolicy, Extended Certificated Authority (ECA), MobiPass and the MobiManager (Figure 1), for a more detailed description of its architecture and functionality, see (Steele, Tao 2006, Tao Steele 2006):

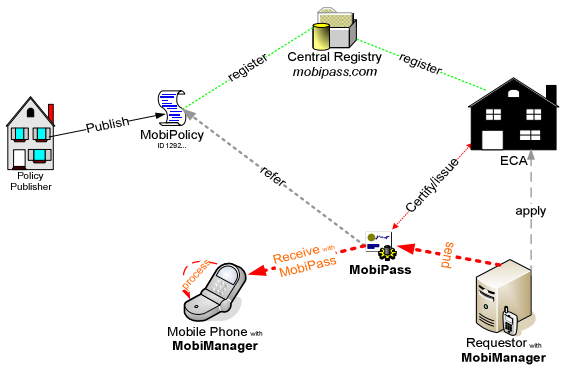


Figure 1: The overall MobiPass architecture

Figure 1 shows that the ECA is an extension of the currently known certificate authority which issues MobiPasses. MobiPass works like a passport in our architecture which is described by XML and complies with the corresponding XML schema, represented in a MobiPolicy. It contains the real data describing a particular service and/or mobile entity in relation to a certain service. Due to the diversity of ubiquitous computing, it is impossible to have one universal specification to model all sorts of services and entities. Therefore a MobiPolicy is introduced to distinguish individual services. It provides a flexible and extensible approach to describe the service and/or mobile entities based on relevant information for this particular service, and MobiPolicy is represented by XML Schema in our architecture. MobiPolicy can be published by any organisations for any services, but the procedure in issuing a corresponding MobiPass is controlled by the ECA, which can also be the same entity as this policy publisher. Moreover, as there is no restriction for any organisation to be an ECA, a non-mandatory Central Registry is introduced to manage all these ECAs. It should be noted that the word *central* in our architecture is only a logical concept. The implementation of a central registry can be totally distributed. The MobiManager is an extra module which is installed on handset devices such as the mobile phone to perform all necessary operations, for example: sending and receiving MobiPass, parsing an incoming MobiPass, helping users to do their preference settings and detecting other surrounding MobiPass devices.

In the MobiPass architecture, multiple ECAs are allowed in the MobiPass with different levels of trustworthiness. Any entity within the MobiPass architecture can act as the ECA to issue certified evaluation results, also, multiple policies are used for different services. A customised policy can be published by any entity to meet the requirement for their particular service.

3 Local Broker Enabled MobiPass Architecture

3.1 Architecture Overview

As described in Section 2, the MobiPass architecture allows previously unknown entities to communicate with each other in a trusted manner. However, in some cases, the performance can be improved if we introduce a Local Broker (LB) into the architecture. As asymmetric key encryption is relatively resource consuming, the performance might be an issue for MobiPass architecture adoption (Diffie 1998, Lenstra & Verheul, 2000). Based on our current research, we have found that in many cases, where mobile entities are previously unaware of each other's existence, there is usually a broker that links all these mobile entities and the broker knows how to deal with each other. Consider the following analogy. Bob organised a party and he invites group of people. However, his guests may not know each other because some of them are Bob's classmates while some of them are his business associates and the rest of them may be his relatives.

Although they do not know each other, one common thing is that they all know Bob and he knows how close he is with each one of them (authentication). If we assume that all guests trust Bob, hence they will assign each other a minimal level of trust, until they have received further information from Bob. The same scenario happens quite often in the area of mobile computing, i.e. the host (referred as the LB in this paper) which provides the service has enough knowledge of participating mobile entities and knows how to assign privileges to different mobile entities with fine grained access control level, and all these mobile entities fully trust the LB (see case study in Section 4). This means that as long as these mobile entities can establish a trusted relationship with the LB, that trust can be expanded across that service network.

Based on our previously published MobiPass architecture, we have developed a variant, the LB enabled MobiPass architecture. This architecture introduces a new element - the LB, and the assumption in this architecture is that the LB is fully trusted during the entire interaction. The LB is the core in this architecture as it does the initial authentication and authorisation and it needs to decide how to assign privileges to different entities. The LB is also responsible for announcing the service so when a MobiPass enabled device enters into the vicinity, it can easily discover the desired service which is being advertised and attempts to initiate communication. To improve the availability, LB can be deployed in a clustering mode which means that multiple LBs can share a single access point address and synchronise the application data in real time. Also, when mobile devices cannot communicate with the LB, devices will try to skip the LB and interact with other transacting parties directly.

An important point is that the LB only works locally, i.e., there is no centralised broker. To establish a trusted link that facilitates interaction between devices, different elements are required to collaborate with each other.

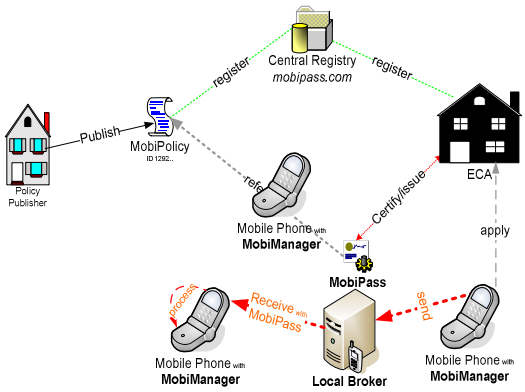


Figure 2: Overview of the Local Broker enabled MobiPass Architecture

Figure 2 demonstrates how the service works. MobiPass enabled devices will keep on discovering the available services.

Compared to the normal MobiPass architecture, there are no changes in how MobiPass is applied by MobiPass users and is granted by ECAs. The users will still go through the normal processes, i.e., first, users are required to find the right MobiPolicy then fill in all necessary information for that particular MobiPolicy. Next, the MobiPolicy will then need to be sent to the ECA and apply to the MobiPass. If all information has been verified as true and genuine, the ECA will issue a MobiPass to this user which contains a valid ECA's digital signature.

The difference with the LB based MobiPass architecture is that rather than having a direct communication with each individual mobile entity, the MobiPass enabled device will talk to the LB instead of directly to the targeted entity. Additionally, the LB can act as a pure "forwarder" or fully on behalf of the involved mobile entities. It should be noted that the LB also holds a special role, it needs to apply a MobiPass from the relevant ECA, which indicates that it is from the MobiPolicy publisher, and the ECA will also verify the relevant documents to make sure that all information in this MobiPass is genuine. It will then sign this LB's MobiPass using the ECA's private key. When the MobiPass is installed in the transaction entity's mobile device, a XML schema based user interface generating system, Xplorer, (Steele et al 2005) will generate preference settings interface based on MobiPolicy's XML schema and users will be required to fill in the options based on the incoming MobiPass's data. For example, in a Mobile social introduction service, users can set his/her preference for their device to only look for software engineers whose age is between 30 and 35. Once the preference is set, users can activate his/her MobiPass enabled service on his/her mobile device. It should also be noted that this LB enabled MobiPass device will keep on discovering any available services which match the MobiPolicy. On the other hand, unlike the MobiPass enabled device, LB enabled MobiPass devices will not advertise themselves; it will only discover the service. The LB is responsible for advertising the available services as all communications are surrounding the LB, so the LB will keep on announcing the services with their corresponding MobiPolicy IDs.

Once the MobiPass enabled device has found the right MobiPolicy ID through the LB, the following actions will take place:

1. The mobile device will ask the LB to send the MobiPass to check whether this LB is a genuine ECA signed Broker.
2. After receiving a MobiPass from a LB, a check will take place to assess whether or not this LB is a Known Local Broker (KLB). A KLB is one that has a public key already existing in the mobile device or its MobiPass can be found in the list containing existing trusted MobiPasses.
3. In the case where a LB is not a KLB, the normal procedure for verifying MobiPasses will be applied onto this broker's MobiPass. Once this has been done, the LB can be confirmed as a true LB for the advertised MobiPolicy.
4. The mobile device will send a list of supported symmetric key algorithms such as DES or BlowFish, it will also send out the MobiPass which contains the user's public key, as well as the ECA's public key and signature to the LB for the random security number.
5. The LB will try to validate the incoming MobiPass by evaluating ECA's signature to see whether this MobiPass is valid, assuming that LB has a very good knowledge of ECAs, especially for ECA's public keys. If the incoming MobiPass is considered as a valid MobiPass and matches the MobiPass's holder, the LB will then choose a supported encryption algorithm, generate a security token (e.g. a large random number), which will then be encrypted by the MobiPass holder's public key that is extracted from the MobiPass. The encrypted security token will then be encrypted by the LB's private key and then sent back to the mobile device.
6. Once the mobile device has received the enhanced MobiPass from the LB, it will decrypt the message by the LB's public key and the MobiPass holder's private key to get the original sender's MobiPass and run the normal MobiPass interaction procedure to establish the trusted interaction.

After this, both LB and the mobile device know the secret key and the secret key is only shared between these two parties. To perform the service smoothly for interacting mobile entities, a session timeout value can be set at the LB to prevent the interacting entities accidentally dropping out of the service. As it is not unusual that the mobile device may roam out of the service vicinity temporarily, once the LB receives the incoming MobiPass and this MobiPass cannot be delivered, this MobiPass will be kept until the session timeout value has been reached. This means that after this time, the LB will consider that this transacting entity has formally quit the session

(service).

As previously mentioned, the LB runs under two modes, they are:

1. Forward only mode
2. Access Level Control (ALC) Mode

Depending on the current condition, the LB will run in either of the modes, the following sub-sections will explain these two modes in detail.

3.2 Forward Only Mode

The Forward Only Mode means that the LB will only pass and forward MobiPasses among mobile entities, no further operations/processing will be made.

One advantage of the Forward Only Mode is that all mobile devices will only communicate to the LB. Once a successful handshake has been initiated, there is no need for the asymmetric encryption anymore in ongoing communications. The mobile device will automatically discover each other, and track down the address. Therefore when they want to start the communication, they only need to forward the MobiPass to the LB, along with the destination address. Once the LB receives the MobiPass, it will decrypt the MobiPass using the secret key which is shared by the sender. When decryption is successfully finished, the LB will encrypt the MobiPass content using another secret key which is shared by the receiver. Therefore during the message transmission stage, only the sender, receiver and LB can read the message, and as the LB is fully trusted, it means that during the transaction of the message, the MobiPass will be safe and there is no need to contact the ECA to ensure that the MobiPass and the content in the MobiPass is authentic. This mode allows mobile devices within the entire network to each do one public key encryption operation and the rest of the operations will be conducted by private key encryption. This greatly improves the performance and eases the communication especially when there are a large number of transacting entities.

When the mobile entity receives the incoming MobiPass, they will run the whole workflow as discussed in our previous paper, please note that even if the MobiPass has successfully been delivered it does not guarantee that a transaction will be conducted. Interactions between devices will only be conducted when the MobiPass matches the receiver's profile to present access control rules.

3.3 Access Level Control (ALC) Mode

ALC mode is a more advanced mode for the LB. Rather than just simply forwarding the incoming MobiPass, it actually runs the authentication and authorisation for the MobiPass.

Once the LB and the mobile entities have finished the handshake, the LB will request the copy of the mobile device's preference for this MobiPass/MobiPolicy profile, and this preference setting will be transmitted by using the shared symmetric key between this mobile device and the LB. In this case, no public key encryption is required anymore as the shared secret key and algorithm is sufficient to identity the sender's ID.

The entire handshake is finished once the LB has successfully received the preference settings. Extending the forward mode, the LB not only forwards the MobiPass, it also runs the preference check on users' behalf every time a transaction occurs. As discussed previously, after the MobiPass has been verified, the LB will request the encrypted preference settings from the mobile device, as the LB has full knowledge of this MobiPolicy, it is very easy for the LB to check all incoming MobiPasses. Under this mode, the LB is responsible to perform the authentication and authorisation for building the trusted interaction between two mobile entities. The steps for communications are explained below:

- Receive the sender's MobiPass.
- Decrypt the incoming MobiPass by using a shared secret key with the sender.
- Detect the designated device from the MobiPass
- Look up the preference settings of this destination device, if the preference setting can not be found, then it will contact the designated device to initiate a handshake, then the symmetric key will be shared and the encrypted preference setting will be acquired.
- If the destination device is no longer in the network, the sender will be notified, otherwise, the LB will extract all the values and compare to the receiver's preferences. If the incoming MobiPass matches the receiver's preference settings, it will be forwarded to start conducting a trusted transaction, otherwise, a request will be sent out.

The ALC mode will greatly reduce the load of the mobile devices, as the computational part has been successfully transferred to the LB. As the LB is not necessarily a mobile device, it can in all probability easily handles the load and perform the authorisation as well. In the next section, a detailed case study will be given to clearly describe the LB enabled MobiPass architecture.

4 Case Study

In this section, a university community based case study will be used to explain how the LB enabled MobiPass architecture can assist in establishing a trusted interaction as the research community is familiar with the university environment. It should be noted that the scope of the LB enabled MobiPass architecture is not limited to the university environment, any environment which requires a trusted interaction between several mobile entities and satisfies the requirements of the LB i.e. transaction entities might not know each other but they all trust the LB, can benefit from this architecture.

These are the facts that exist in most public universities:

- There are a large number of students in the university; the number of student can exceed 100,000, and many different units coexists in the university, such as faculties, departments, service units, student unions and clubs.
- Most students only know a limited number of other students in the university. However

collaborations are often required, even though students/staff do not know each other.
- Every staff/student is supposed to trust the administration unit in the university.

From these facts, we can derive that there is a demand for a trusted interaction and it is not an easy task to implement such an interaction within the university as there might be a large number of staffs and students, all with a different background e.g. language and culture. Moreover, most of them do not have a previous knowledge of each other. For example, a group of students from different faculties doing some outdoor activities together, or they are looking for a flat mate to share accommodation with. A trusted interaction is required for transactions within all the above mentioned cases. We will now use the Accommodation Finder Service (AFS) as an example to demonstrate how the LB enabled MobiPass architecture works.

As there are many rural, inter-state and international students in the university, it is necessary for them to find their own accommodation as it might not be financially feasible for them to travel to the university from home everyday, so some students will go and find others to share accommodation. Also, due to security concerns, students like to share with other students from the same university; some students even like to share accommodation with others who have the same background or interest. Currently the main approach which has been used in many universities to find a flat mate is to read notes or advertisements posted on bulletin boards, this way can be dangerous as by just referring to the information given out in the post, there is no way in telling whether the information is true. Also, as this is not a real time interaction, a student will need to arrange and meet with a potential flat mate somewhere else. This kind of appointment can be dangerous, especially for female students. By using the LB enabled MobiPass architecture, it is very easy to enable a trusted interaction for the AFS.

In this case, the ECA and MobiPass publisher will be the Student Service Union (SSU) and as students currently trust the SSU; we can assume that trust will extend to the SSU published ECA and MobiPass. SSU can provide a online form which has an equivalent schema as the AFS MobiPolicy, and students can carry out and apply for their AFS MobiPass from this portal. Students will only be required to fill in extra information such as their interests and hobbies, as the SSU already has part of the student's personal information which has already been authenticated, such as their real name, gender, age, major and nationality. The extra information will only act as a supplement to help students to find flat mates who are more compatible with them and therefore certification is not required. Once they have filled in the forms, the SSU will generate the MobiPass for the AFS for this student.

For instance, Alice is a 20 years old first year international student who is studying computing science, and she is currently looking for a flat mate. Alice would like to share accommodation with another female international student of a similar age and background because this makes her feels comfortable and safe. Therefore her preferred flat mate will be a female student, aged between 20 to 25 years old, and can speak her language. Alice does not want to use the traditional approach to find flat mates as Alice can only gather potential flat mate's information by reading posts and there is no way to tell whether this information is true or false. Also, after she has contacted the poster, she might go somewhere else with this potential flat mate to find accommodations. This can happen at night and Alice feels that it is dangerous to meet with a stranger in an unknown location. So she signs onto the SSU portal site and applies for a MobiPass for the AFS. To ensure the level of security, Alice must hold her public/private key pair before applying for the MobiPass from the SSU. Once she has signed in, she would found that most information about her has already been filled and cannot be modified; only the self explanation and descriptions are left for her to fill. At the same time, Alice's public key for the AFS is uploaded to the SSU. Once submitting the form and reviewed by the SSU staffs, Alice will receive a MobiPass for the AFS from SSU. The message snippet is shown in Figure 3.

```
<MobiPass>
  <meta>
    <digestValue>RjzP...DGY8=</digestValue>
    <signatureValue>=</signatureValue>
  </meta>
  <certified>
     <expired>2007-08-05</expired>
    <issuer>
      <ECA>
        <ECA-ID>124..626</ECA-ID>
          <ECA-name>AFS, Univ of Techo, Sydney </ECA-name>
<publicKey>https://ssu.mobipass.uts.edu.au/afs.pub.key</publicKey>
      </ECA>
        <policy>
          <policy-ID>11...34</policy-ID>
          <description>....</description>

<schemaLocation>https://ssu.mobipass.uts.edu.au/afs.schema.xsd</sche
maLocation>
      </policy>
    </issuer>
    <studentInfo>
      <publicKeyOfHolder>Daz==.z==</publicKeyOfHolder>
      <gender>female</gender>
      <age-range>20-25</age-range>
      <department>computer science</department>
      <faculty>information technology</faculty>
      <nationality>Chinese</nationality>
          …
    </studentInfo>
  </certified>
  <nonCertified>
     <selfDescription><![CDATA[… easy going, nice person!....
]]></selfDescription>
<interests>
        <element>fishing</element>
        <element>reading</element>
        <element>...</element>
      </interests>
<smoker>false<smoker>
<hasPet>false<hasPet>
  </nonCertified>
  <timestamp>
```

```
    <notBefore>1132622517640</notBefore>
    <notAfter>1132622519640</notAfter>
    <timestampSignatureValue>skz...==z</timestampSignatureValue>
  </timestamp>
</MobiPass>
```

Figure 3: MobiPass Message Snippet for AFS

After receiving this AFS MobiPass, Alice imports this MobiPass to her mobile phone and tries to setup the service correctly. We can assume that Alice already has SSU's public key in her mobile phone, and the MobiPolicy for this AFS has been downloaded onto her mobile phone. So Alice runs her MobPolicy setup to load her AFS service, and fill in all other criteria for this service. Such as:

- Gender: Female
- Flat mate Age Range: 20-25
- Nationality: {Chinese, Korean, Japanese, Australian}
- Flat mate Major: Accounting, Business, Music
- List of Suburbs, which are within walking distance to the university
- Monthly rental budget, for Alice, the limit is $150 per week.
- Furnish – fully furnished etc

The service is activated once the setup is finished. So when Alice walks into the campus, her mobile phone will try to find an AFS LBs to run the service. For example, the AFS LB reception might cover the central common areas within the university, and the LB will run the service in ALC mode. This means that when Alice enters the campus, the AFS service will be up and running on her mobile phone, then her MobiPass application will try to contact the LB for the AFS service. When Alice's mobile device has found the SSU's LB, it will then try to authenticate the LB; initiate a handshake with the local SSU for exchanging the symmetric key. As the local SSU is running in ALC mode, the SSU's LB will also be asking for Bob's AFS service settings, so Alice's mobile device will send his settings to the SSU's LB. The SSU's LB will then act on Alice's behalf and announces to the entire wireless network that a new member has joined the network and this new member is looking for a flat mate. After receiving the message, students who are using the same service might try to contact Alice by their MobiPass through the LB, and the SSU LB will try to authenticate the incoming MobiPass for Alice, i.e, whether this sender has a valid MobiPass e.g. a student from the university. If the MobiPass is valid, the LB will try to match their profiles and if it matches Alice's criteria, she will receive a notification that there are people around who are interested to share an accommodation with her.

This notification by her MobiPass-enabled device will allow Alice to meet with potential matching students within a very short time in the university common area, so that they can speak face-to-face, therefore allowing Alice to make a final decision on whether the potential student is a match. In this way, trusted interaction for mobile devices supported by the MobiPass architecture can provide greater immediacy and functionality than other electronic interactions.

This case study has described the steps taken in applying the LB based MobiPass architecture to the AFS, the MobiPass architecture, in this example, provides an excellent platform for university students to find their accommodation and flat mates securely and efficiently. By using the MobiPass architecture, students can very easily distinguish potential flat mates. The scope of potential flat mates is limited to university students with matching profiles. Also, the communication can happen in real time and once they find each other by mobile phone, they can start meeting immediately, such as in the university's common area where the MobiPass-based interaction initially occurs. Students are not required to make appointments and meet somewhere which might be unfamiliar and potentially dangerous to them. Also the architecture is very open and flexible; it is easy to apply this architecture to a more mission critical service to ensure that interactions will take place in a trusted manner.

5 Related Work

This research outcome is based on our previous research on ubiquitous and mobile computing, the MobiPass architecture (Steele, Tao 2006, Tao, Steele 2006). The goal of this research is to provide a highly effective approach to build a trusted interaction between different entities within an open and dynamic environment. There are also other researchers that have focused their efforts in addressing this issue.

Kagal, Finin and Joshi (Kagal, Finin, Joshi 2002) proposed the Centaurus system which provides a fine grained access control in their Smart Office ubiquitous computing scenario, the system utilises the distributed trust approach and extends the Role Based Access Control(RBAC) to allow foreign users from another security domain to be granted the proper privileges in order to gets access. Based on their implementation, Hong and Landay (2004) propose the architecture to perform the authentication and authorisation in ubiquitous computing by assigning tags to pieces of information; information is associated with a policy and indicated by the tag. Park and Sandhu (1999) proposed a concept named smart certificate for improving scalability in web servers, which has some interest to our work. The smart certificate is an extended version of X.509 certificate with several remarkable features. These previous research works are more focused on authentication which can help interacting entities to identify the transacting parties; however, fine grained access control is not covered comprehensively, which, is a very important aspect in mobile computing.

6 Future Work and Conclusion

To build a trusted environment in mobile computing, the MobiPass architecture is introduced to allow mobile devices to be recognised by only presenting their MobiPass and also it allows one entity to judge other entities by examining their respective MobiPasses. However, for performance considerations, a variant that introduces the LB to reduce the load for each mobile device has been introduced in this paper. The LB enabled MobiPass architecture works under the condition that a group of mobile devices do not know each other, but they

all have a solid relationship with the LB. According to our research, there are a large number of scenarios in which the device has such good knowledge of the LB. Our future work will focus on extending the LB enabled MobiPass architecture to enable arbitrary MobiPass processing in real time, i.e. do not need to have manually imported the ECA's public key, and proposing a good mechanism for allowing to reuse a MobiPolicy by multiple service providers. Efforts will also be made to improve our service discovery protocol, i.e., how different entities can discover each other and how to negotiate and send the MobiPass during the discovery. Our ongoing research will be to refine the MobiPolicy and to make the MobiPass architecture generic enough to be pluggable into most trustworthiness mobile systems.

References

Steele, R., Tao, W. (2006) MobiPass: A Passport for Mobile Business. *Personal and Ubiquitous Computing*, Springer, 11 (3): pp.157-169

Tao, W., Steele, R. (2006) Mobile Trust Via Extended Digital Certificates, *Proc of The 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing (DASC 06)*, pp. 284-292.

Satyanarayanan, M. (2001), Pervasive computing: vision and challenges, *IEEE Wireless Communications, 8 (4)*, 10-17

Ranganathan, K. (2004), Trustworthy Pervasive Computing: The Hard Security Problems, *Proc of the 2nd IEEE Annual Conference on Pervasive Computing and Communications Workshops* 119

Diffie, W. (1998), The first ten years of public-key cryptography, *Proc of IEEE, 76 (5)*, 560- 577

Steele R, Gardner W, Rajugan R, Dillon TS (2005) A design methodology for user access control (UAC) middleware. *Proc of the 2005 IEEE international conference on e-technology, e-commerce and e-service (EEE'05)*, pp 385–390

Kagal L, Finin T, Joshi A (2002) A security architecture based on trust management for pervasive computing systems". *Grace Hopper Celebration of Women in Computing,* October 2002

Hong J, Landay J (2004) An architecture for privacy-sensitive ubiquitous computing" *Proc of the 2nd international conference on mobile systems, applications, and services*, pp 177–189

Park J.S, Sandhu R (1999) RBAC on the Web by smart certificates *Procof the 4th ACM workshop on role-based access control. ACM Press,* New York, pp 1–9

HOVER: Hybrid On-demand Distance Vector Routing for Wireless Mesh Networks

Stephan Mir†, Asad Amir Pirzada and Marius Portmann‡

Queensland Research Laboratory,
National ICT Australia Limited,
Brisbane, QLD 4000, Australia.
{stephan.mir,asad.pirzada,marius.portmann}@nicta.com.au

Abstract

Hybrid Wireless Mesh Networks are a combination of mobile ad hoc networks and infrastructure wireless mesh networks, consisting of two types of nodes: mobile Mesh Clients and static Mesh Routers. Mesh Routers, which are typically equipped with multiple radios, provide a wireless multi-hop backhaul. The resource constrained Mesh Clients also participate in the routing and forwarding of packets to extend the reach of the network. Current ad-hoc routing protocols have been designed for relatively homogeneous networks and do not perform well in Hybrid Wireless Mesh Networks. In this paper, we present HOVER (Hybrid On-demand Distance Vector Routing), a modified version of the AODV routing protocol, that achieves significant performance improvements in terms of packet delivery and latency in Hybrid Wireless Mesh Networks. Our modifications include a link quality estimation technique based on HELLO packets, a new routing metric that differentiates between node types, and a channel selection scheme that minimises interference in multi-radio mesh networks. We present an evaluation of our improvements via extensive simulations. We further show the practicality of the protocol through prototype implementation and provide measurement results obtained from our test-bed*.

Keywords: Multi-radio, routing, mesh, wireless, network

1 Introduction

Wireless Mesh Networks (WMN) have recently gained considerable popularity due to their self-healing, self-configuring and self-optimising capabilities. The low cost of commodity IEEE 802.11 wireless hardware, on which most WMNs are based, further adds to the appeal of the technology. WMNs offer an attractive platform for a wide range of applications, such as public safety and emergency response communications, intelligent transportation systems, and community networks.

A WMN consist of two types of wireless nodes: Mesh Routers and Mesh Clients. The Mesh Routers have improved computational, communication and power resources as compared to Mesh Clients. Mesh Routers are generally static and form the multi-hop backhaul network. A subset of Mesh Routers can also provide gateway functionality and connect the WMN to external networks. Mesh Routers are also typically equipped with multiple wireless network interfaces and are, therefore, able to establish high capacity connections by using multiple orthogonal channels. Mesh Clients are mobile devices, which take advantage of the existing communication infrastructure provided by the Mesh Routers.

WMNs can be divided into three main types (Akyildiz & Wang 2005): Infrastructure, Client, and Hybrid. In an Infrastructure WMN, Mesh Clients gain access to each other or to the backhaul network through Mesh Routers and are not actively involved in routing and forwarding of packets. Thus, all Mesh Clients gain access to Mesh Routers via a single wireless hop. In Client WMNs, Mesh Clients communicate with each other directly, without involving any Mesh Routers. A Client WMN is essentially a pure multi-hop mobile ad-hoc wireless network (Pirzada & McDonald 2004).

A Hybrid WMN combines the connectivity pattern of both the Infrastructure and Client WMNs. In these networks, both Mesh Clients and Mesh Routers are actively involved in routing and forwarding of packets, and Mesh Clients can access the wireless backhaul network via multiple client hops.

A typical scenario where a Hybrid WMN might be employed is in emergency response and disaster recovery situations, where traditional communications infrastructure might not be available. In such a case, a hybrid WMN can provide a so-called incident area network, as illustrated in Figure 1.

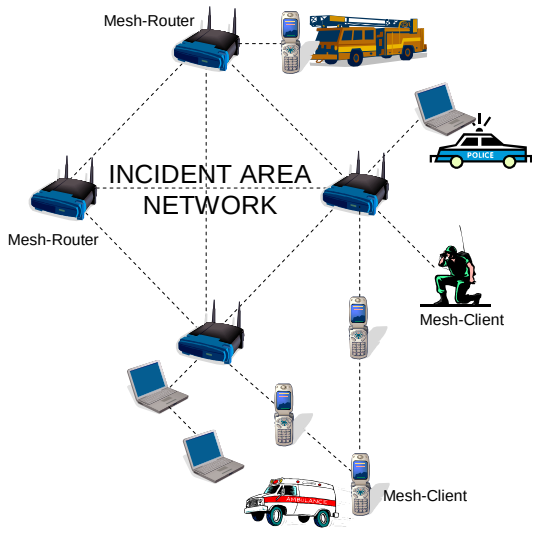


Figure 1: Hybrid Wireless Mesh Network

†The author is also affiliated with Télécom Paris, ENST, Paris, France and Ecole Polytechnique, Palaiseau, France.

‡The author is also affiliated with the School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane, QLD 4072, Australia.

*The research work presented here is in continuation with our primary concept paper published in the Proceedings of the Thirtieth Australasian Computer Science Conference (Pirzada, Portmann & Indulska 2007).

Current WMNs suffer from the problem of performance degradation with increasing number of wireless hops. One of the reasons is the limitation of the IEEE 802.11 MAC layer based on CSMA/CA, which has not been designed for multi-hop networks. A major problem is co-channel interference, where nodes within interference range are transmitting simultaneously on the same channel, resulting in collisions, reduced throughput and increased communication delays (Gupta & Kumar 2000). A further limiting factor is the half-duplex nature of IEEE 802.11 network interface cards, which does not allow simultaneous sending and receiving.

Even though some of these problems can be mitigated by providing nodes with multiple network interfaces (Raniwala & Chiueh 2005) (Kyasanur & Vaidya 2006), the problem of finding optimal routes in WMNs, and in Hybrid WMNs in particular, is largely an open research issue. Challenges are mobility, heterogeneity and the problem of channel selection with the goal of minimising interference.

The Ad-hoc On-demand Distance Vector (AODV) protocol (Perkins, Royer & Das 2003) is one of the predominant reactive ad-hoc routing protocols. AODV was originally developed for homogenous mobile ad-hoc networks, where nodes typically have a single wireless network interface and have comparable computational and communication resources. Consequently, AODV has some shortcomings when applied to Hybrid WMNs. First of all, with hop-count as its routing metric, it lacks the ability to differentiate between node types, i.e. Mesh Routers and Mesh Clients. Thus it is unable to exploit the heterogeneity in the network. Furthermore, AODV is not aware of the wireless channels used for the network interfaces and, therefore, it cannot minimise co-channel interference and maximise the use of multiple orthogonal channels between node pairs.

In this paper, we present three modifications to AODV to address these shortcomings and to improve its performance in Hybrid WMNs. HOVER (Hybrid On-demand Distance Vector Routing), our modified version of AODV, outperforms standard multi-radio AODV by a considerable margin, as we will show in Section 4 and 5. Our proposed modifications, and also our key contributions, are as follows:

- We propose a modification to AODV's route discovery mechanism that maximises the involvement of Mesh Routers in the establishment of end-to-end paths. It is achieved by introducing a new routing metric that differentiates between Mesh Routers and Mesh Clients.
- In a multi-radio WMN, neighbouring nodes can be connected via multiple links on orthogonal channels. We integrated a channel selection scheme into AODV's route discover mechanism that can select the"best" of these available links for each hop of the path.
- We implemented a link quality estimation scheme based on HELLO packets in AODV. HELLO packets are broadcast regularly by each node to monitor link connectivity and to detect link breaks. Hence, no additional overhead is introduced in the network.

The rest of the paper is organised as follows. Section 2 discusses relevant related work. The HOVER protocol is discussed in detail in Section 3. Simulation results and their analysis are discussed in Section 4. A prototype implementation of HOVER and its evaluation is presented in Section 5. Section 6 concludes the paper.

2 Related Work

AODV-ST (Ramachandran, Buddhikot, Chandranmenon, Miller, Belding-Royer & Almeroth 2005) is a Hybrid routing protocol developed specifically for infrastructure mesh networks. The protocol has been designed with the aim of providing Internet access to Mesh Clients with the help of one or more gateways. AODV-ST uses a proactive strategy to discover routes between the Mesh Routers and the gateways, and a reactive strategy to find routes between Mesh Routers. In the proactive case, the gateways periodically broadcast special Route Request packets to initiate the creation of spanning trees. All subsequent Route Request packets with a better routing metric are used to update the existing reverse route to the gateway. AODV-ST uses the Expected Transmission Time (ETT) (Draves, Padhye & Zill 2004) routing metric, which measures the expected time needed to successfully transmit a fixed-size packet on a link. The ETT of a link is computed using the Expected Transmission Count (ETX), bandwidth and packet loss. AODV-ST has primarily been designed for single-radio wireless nodes and hence cannot exploit the full potential of multi-radio nodes in the network. Neither does it differentiate between different types of nodes in Hybrid WMNs.

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 implements a routing protocol and supports a fully distributed channel assignment algorithm, which can dynamically adapt to varying traffic loads. Hyacinth's channel assignment algorithm breaks a single-channel collision domain into multiple collision domains, each operating on a different frequency.

The Multi-Radio Link Quality Source Routing (MR-LQSR) (Draves et al. 2004) protocol has been developed for static community wireless networks. The protocol works in conjunction with the Mesh Connectivity Layer (MCL). The protocol identifies all nodes in the wireless mesh network and assigns weights to all possible links. The link information including channel assignment, bandwidth and loss rates are propagated to all nodes in the network to compute the Weighted Cumulative Expected Transmission Time (WCETT), which is a routing metric that also takes channel diversity into account.

The Multi-Channel Routing (MCR) protocol (Kyasanur & Vaidya 2006) has been developed for dynamic WMNs where nodes have multiple wireless interfaces and each interface supports 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. Switching is carried out depending upon the maximum number of data packets queued for a single channel. 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-count. 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 is considered to have lower diversity cost. The switching cost is used to minimise the frequent switching of wireless interfaces.

Some of the works discussed use dynamic channel allocation and switching mechanisms to improve the routing performance of the network. However, the accurate and synchronised execution of these mechanisms in a mobile network requires the availability of a virtual switching protocol and incurs switching delays (Chandra & Bahl 2004, Draves et al. 2004).

Our protocol avoids this complexity by assuming a static allocation of channels to interfaces, while still achieving significant performance improvements.

Some related work introduce new routing metrics to improve the quality of end-to-end paths. However, to the best of our knowledge, HOVER is the first protocol that specifically addresses the problem of routing in Hybrid WMNs by differentiating between different node types. A further contribution of this paper is the integration of an intelligent channel selection mechanism with the route discovery mechanism of a reactive routing protocol.

3 Hybrid On-Demand Distance Vector Routing (HOVER) Protocol

Our contributions presented in this paper consist of a number of modifications to AODV. Therefore, we first provide a brief overview of AODV's route discovery mechanism before discussing our proposed extensions.

3.1 AODV

The AODV routing protocol is 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 (Johnson, Maltz & Hu 2003) and hop-to-hop routing vectors from the Destination-Sequenced Distance-Vector (DSDV) routing protocol (Perkins & Bhagwat 1994). 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) with a unique ID field. Each recipient of the RREQ, which has not seen a RREQ with the same source IP and ID pair or does not maintain a fresher (with larger sequence number) route to the destination, rebroadcasts the same packet. Such intermediate nodes also create a reverse route to the source node.

When the RREQ reaches the destination or an intermediate node, which has a fresher route to the destination, a Route Reply packet (RREP) is generated and unicast back to the originator of the RREQ. Each RREP contains the destination sequence number, the source and the destination IP addresses, route lifetime, together with hop-count and control flags. Each intermediate node that receives the RREP increments the hop-count and establishes a forward route to the source of the RREP. The RREP finally reaches the originator of the RREQ by traversing reverse routes that were established while forwarding the corresponding RREQ.

To maintain connectivity information, nodes can either use link-layer feedback or periodic Hello packets to detect link breaks to immediate neighbours. In case a link break is detected for a next hop of an active route, a Route Error (RERR) packet is sent to the active neighbours that were using that particular link.

When using AODV on a multi-radio node each RREQ is broadcast on all interfaces. Intermediate nodes with one or more interfaces operating on a common 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 RREQ received by the destination or any intermediary node is selected and all other RREQs belonging to the same route discovery are discarded. The RREP is generated in response to the selected RREQ and is sent back to the source node on the established reverse route (Pirzada, Portmann & Indulska 2006). We refer the standard AODV protocol with support for multiple radios as AODV-MR in this paper.

3.2 HOVER

HOVER uses the same basic route discovery mechanism as that of AODV-MR. However, in order to guarantee that routes are preferentially established via Mesh Routers, and to provide optimal link selection when multiple links exist between immediate neighbours, it implements the following three additional mechanisms:

- Node-type aware routing
- Link quality estimation
- Optimal link selection

3.2.1 Node-type aware routing

Hop-count is the default routing metric used by AODV-MR. However, this metric does not guarantee the selection of optimal paths. A longer path consisting of high quality links can perform much better than a shorter path consisting of low quality links. Since Mesh Routers are more static, have multiple radios and are typically equipped with higher gain antennas, they are expected to maintain higher quality links with their neighbours than Mesh Clients. A longer path comprised of Mesh Routers can, therefore, be expected to perform better than a shorter path comprised only of Mesh Clients. The basic idea is to preferentially involve Mesh Routers in the creation of end-to-end paths. This not only improves the performance of the path, but also minimises unnecessary draining of batteries belonging to resource constrained mobile devices.

We therefore introduce a new path cost routing metric, which is computed as follows:

$$Cost = \quad MR\_COUNT \times MR\_COST + MC\_COUNT \times MC\_COST$$

The parameter MR_COUNT represents the number of Mesh Routers in the path and MC_COUNT represents the number of Mesh Clients. MR_COST and MC_COST are the relative weights (or costs) associated with each type of node. In our implementation, we set the $MR\_COST = 1$ and $MC\_COST = 4$ implying that the Mesh Client traversal cost is four times higher than the Mesh Router traversal cost. These costs were determined through a series of simulations and actual tests.

The path cost is computed by accumulating the individual link costs during the route discovery process. The path cost metric is stored as a 8-bit field in the extension header for RREQs and RREPs.

During the propagation of RREQs, this cost field is used for the creation of optimal reverse routes from destination and intermediary nodes back to the originator of the RREQ. Similarly, while forwarding the RREPs, the cost field is used for creating optimal forward routes. This mechanism is explained in detail in Section 3.3.

3.2.2 Link quality estimation

Each node running HOVER maintains a set of links to its adjacent nodes. Since we consider multi-radio nodes, multiple links can exist between a pair of nodes. The maximum number of links is limited

by the number of available interfaces on the nodes. HOVER keeps track of the quality of these links by listening to Hello packets from its neighbours. When a node receives a Hello packet from a neighbour it validates the link, which is identified by the receiving interface and the IP address of the sender.

When no Hello packets are received for a given period of time ($ALLOWED\_HELLO\_LOSS \times HELLO\_INTERVAL$), the corresponding link is marked as invalid. If the invalidated link is part of an active route, HOVER simply switches the route to use another valid link to the same neighbour. This repair mechanism is extremely efficient and quick, since it is done locally and does not involve sending any routing control packets, in contrast to AODV-MR's local route repair mechanism.

To keep track of the quality of the links between two nodes, we count the number of Hello packets that are received and lost over a period of time. In our implementation, we set this time window to $10 \times HELLO\_INTERVAL$ seconds. Individual Hello counters are maintained for each neighbour and interface pair. These counters allow us to find the parameters d_r and d_f, which are the delivery ratios of Hello packets for the reverse and forward direction of a link respectively.

The parameter d_r is simply the number of Hello packets that were actually received in a given time window, divided by the number of expected Hello packets. This parameter, therefore, indicates the link quality for incoming traffic, i.e. the reverse route direction. Since wireless links are often asymmetric, this quality is not necessarily the same for traffic flowing in the other direction.

The parameter d_r is communicated back to the corresponding neighbour by piggybacking it on the RREPs. For the receiving node, the received parameter d_r corresponds to its parameter d_f, i.e. the delivery ratio on the forward path. With both of these parameters, a node can compute the Expected Transmission Count (ETX) (Couto, Aguayo, Bicket & Morris 2005) parameter.

$$ETX = \frac{1}{d_f \times d_r}$$

We use ETX as a simple link quality metric for all the links that a node maintains with its immediate neighbours. This link quality metric can be used to select the best link between two nodes during route creation.

3.2.3 Optimal link selection

During the route discovery phase, an end-to-end path is established through the concatenation of individual links. The nodes involved in the path are determined by our node-type aware routing metric. However, if multiple links exist between two neighbouring nodes in the path, we need to select one of these links to be used for the route. Our goal is to choose the link that provides the best performance for the path.

During each path discovery process, every node considers the quality of its links to the relevant neighbour and computes a preference or grading value for each link. In addition to the ETX link quality metric, a node can take into consideration other factors such as channel diversity or current channel use, e.g. determined by the number of active flows using that channel (Ko, Padhye, Misra & Rubenstein 2005).

When the RREPs are forwarded, each node also recommends the grading of each link on the reverse path. The grading value is piggybacked onto the RREPs and helps the RREP recipient in determining the optimal link to be used for communication with the RREP sender.

To convey the path cost and link grading, we add an extension header (shown in Fig. 2) to all RREQs and RREPs.

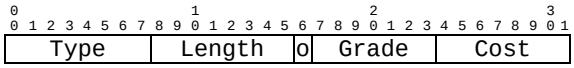


Figure 2: HOVER Packet Extension

The extension header further contains an optimisation flag (o-flag), which will be discussed below. Path cost and link grades are expressed as 8-bit and 7-bit fields respectively. The o-flag is used to differentiate between the non-optimal and optimal RREPs. The extension is compliant to the IETF RFC (Perkins et al. 2003) and will be ignored by nodes that do not understand the extension. Alternately, the existing reserved fields in the RREQs and RREPs can be used to convey the cost, grading and o-flag. The cost field is only used for RREQs, while all three fields are used for RREPs.

3.3 Path Discovery in HOVER

The basic idea of HOVER's path discovery mechanism is to establish a potentially non-optimal path as quickly as possible, in order to minimise the path creation delay. The path discovery mechanism is not finished at this stage but continues to search for a more optimal path. When the optimal path has been found, the route is switched over to the new one.

This is implemented as a two phase optimisation process. The first phase is carried out during the propagation of the RREQs, while the second phase is carried out during the propagation of the RREPs. Phase I and II of this process are depicted in Figures 3 and 4 respectively. The behaviour of AODV-MR is indicated with solid lines, while HOVER extensions are shown via dashed lines.

During Phase I, each intermediary node that receives a RREQ first checks whether it is a duplicate or not. If this is not the case and the packet is the first RREQ with a particular ID, the node stores information about this RREQ, including its cost metric and its ID. A reverse route is then created to the source of the RREQ.

If the RREQ is a duplicate, it is accepted only if it has a lower path cost. Thus, an intermediate node may forward multiple copies of the same RREQ if the path cost of subsequently received RREQs is lower than the previously received copy. This is in contrast to AODV-MR, where only the initially received RREQ is forwarded.

If an intermediate node has a route to the destination, it can send only a single RREP to the first received RREQ with the same ID. This condition is imposed in order to avoid intermediate nodes from replying to all copies of the same RREQ, which would result in significant overhead. For all subsequent copies of the same RREQ, intermediate nodes simply set the 'Destination Only'[1] flag in the RREQ and forward it (Perkins & Royer 1999).

In order to establish an initial route as quickly as possible, the destination responds immediately to the first received RREQ. To indicate that this is a temporary, and possibly non-optimal route, the o-flag in the RREP header is set to false.

[1]The 'Destination Only' flag indicates that only the destination can respond to this RREQ.

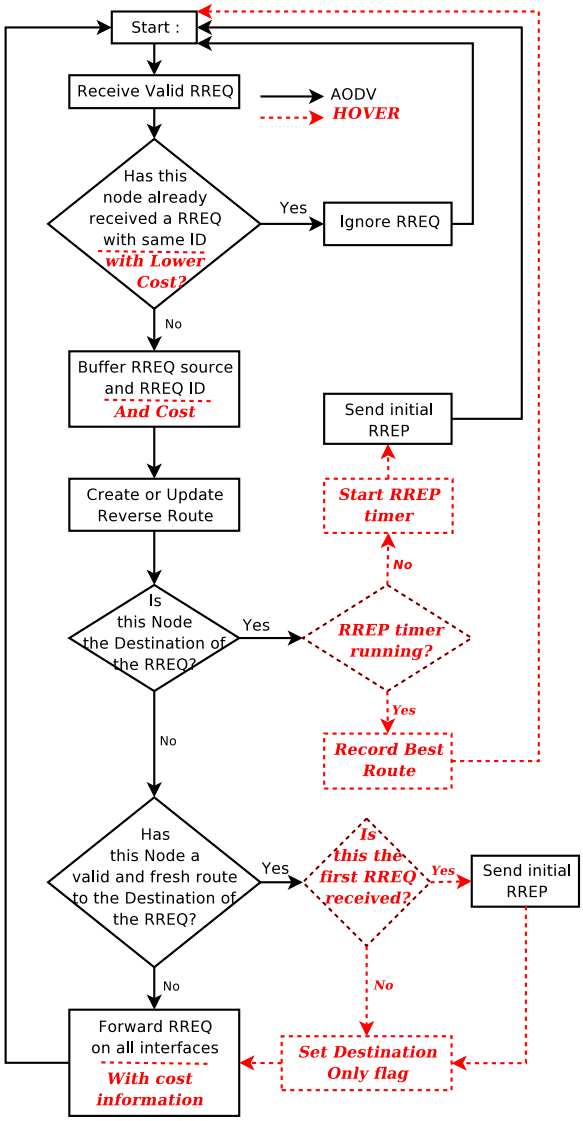


Figure 3: Phase I Flow Chart

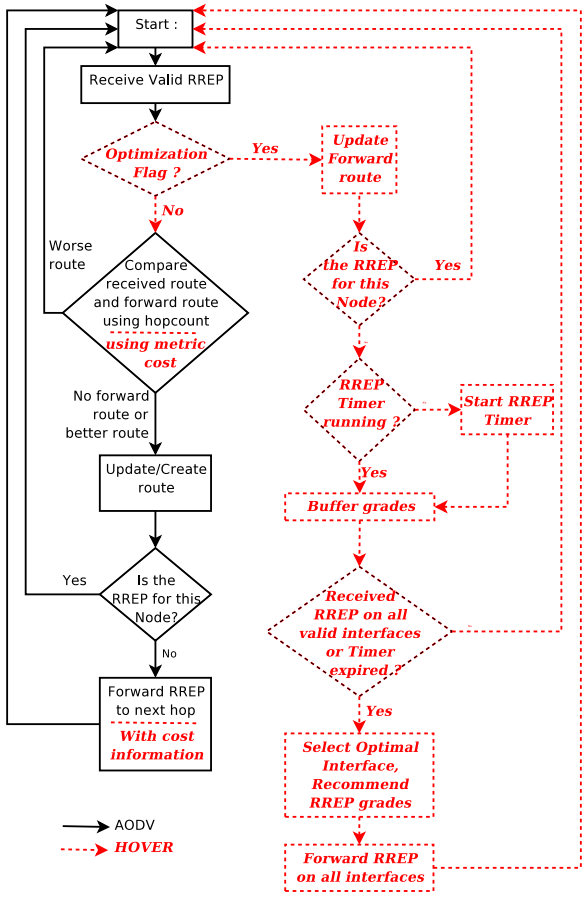


Figure 4: Phase II Flow Chart

This non-optimal RREP is used to establish a path from the source to the destination, which may be sub-optimal, both in terms of Mesh Client involvement and link selection. This essentially replicates the behaviour of AODV-MR.

In HOVER, once an initial non-optimal RREP has been sent, the destination node starts a RREQ optimisation timer. Until the timer expires, the destination node continues to receive and buffer RREQs with the same ID. When the timer expires, the destination node selects the optimal RREQ, i.e. the one with the lowest path cost according to our metric, and replies with a corresponding optimal RREP.

Both the non-optimal and the optimal RREP follow the Phase II process. In contrast to the non-optimal RREP, the optimal RREP is sent via all wireless interfaces. Before sending the optimal RREP, the responding node sets the o-flag to true and initialises the path cost field to zero. The node further sets the link priority or link grading, based on the link quality metric or other factors such as channel diversity. This allows the node to recommend a link for the forward route to the next node.

Sending of multiple RREPs for a single route discovery is one of the key differences from AODV-MR, where the RREP is sent only on the single wireless interface on which the first RREQ was received. The primary reason for sending multiple RREPs is to give the receiving node the opportunity to select the optimal link for the forward route. This also avoids the problem of uni-directional links being created as a result of Hello packets having a longer range than the RREPs (Chakeres & Belding-Royer 2002), since they are sent via broadcast at a lower rate than unicast packets.

The non-optimal RREP is treated as a normal RREP, with the only difference being that the path cost field is updated according to the same rule used during the forwarding of RREQs. If the corresponding forward route already exists at the node that received the RREP, the cost of the existing route is compared with the cost field of the RREP. In case the path cost value in the RREP is lower than the cost of the existing forward route, the route is updated and the RREP is forwarded. Otherwise, the RREP with a higher cost is simply discarded.

The optimal RREP is transmitted on all wireless interfaces by the destination or the intermediate nodes. Upon receipt of a RREP, all nodes first check the status of the o-flag. If the flag is set the RREP is buffered and a RREP optimisation timer is started. The node continues to receive and buffer RREPs (including link grades) with the same ID and a set o-flag until the RREPs have been received via all interfaces or the optimisation timer has expired.

If multiple optimal RREPs have been received, the receiving node needs to select one for creating or updating the forward route to the sender of the RREP. The selection of RREP is based on the link grading and local link information, which includes link usage and channel diversity. The main advantage of sending the link priorities or grades in the optimal RREP is that it provides the RREP recipient with link quality information from the perspective of the RREP sender.

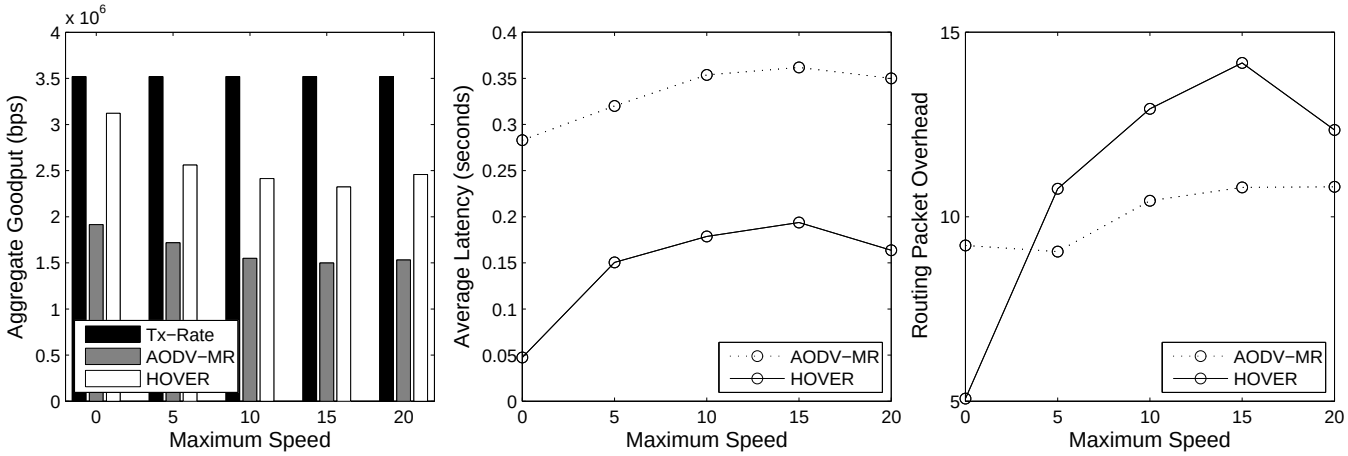


Figure 5: Test 1 Results - Varying the Mesh Client speeds

This process continues until the optimal RREP is received by the source that initiated the route discovery. The source now switches from the earlier created non-optimal route to the newly formed optimised route.

4 Simulation Results and Analysis

4.1 Simulation Environment

We evaluated the efficiency of the HOVER protocol through extensive simulations in NS-2 (NS 1989), using the Extended Network Simulator (ENS) extensions (Raman & Chebrolu 2005). A dense WMN covering an area of 1 square km is established using 25 Mesh Routers arranged in a regular 5x5 grid, with a distance of 176 meters between immediate neighbours. The network further consists of 50 mobile Mesh Clients. Concurrent UDP flows are established between randomly selected source and destination Mesh Client pairs.

The following two tests were conducted to evaluate the performance of the HOVER protocol under varying mobility and traffic load conditions:

- Test 1: Varying the Mesh Client speeds
- Test 2: Varying the traffic load

Table 1: Simulation Parameters

Examined protocols	AODV-MR & HOVER
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	20 m/s
Transmission range	250 m
Number of Connections	30
Traffic type	CBR (UDP)
Packet Size	512 bytes
Packet Rate	32 pkts/sec
Transmission Rate	128 kbps/flow
Number of Mesh Routers	25
Number of 802.11b radios in Mesh Router[2]	6
Number of Mesh Clients	50
Number of 802.11b radios in Mesh Client	1
RREQ Optimisation Timer	1 second
RREP Optimisation Timer	20 milliseconds

The performance metrics are obtained by ensemble averaging the results from over 50 individual simulation runs for each test (Pirzada, McDonald & Datta 2006). The parameters common to the two tests are listed in Table 1.

The simulations provide the following performance metrics:

- Aggregate Goodput: The total number of application layer data bits successfully transmitted in the network per second.
- Average Latency: The mean time (in seconds) taken by the data packets to reach their respective destinations.
- Routing Packet Overhead: The ratio of the total number of control packets generated to the total number of data packets that are successfully received.

4.2 Test 1 : Varying the Mesh Client Speeds

In Test 1, we varied the speed of the Mesh Clients from 0 to 20 m/s. The results of Test 1 are shown in Fig. 5.

AODV-MR endeavours to create the shortest path (in terms of the number of hops) between a source and destination node pair. Thus it essentially ignores whether a node is a Mesh Router or a Mesh Client and focuses on rapid route creation. The rationale behind the rapid route creation is that the quickest path is most likely the optimal path. Thus paths created by AODV-MR may or may not consist of Mesh Routers. Thus once the traffic starts to flow over the paths, we observe a high packet loss due primarily to the interference observed in the dense network. As the number of flows is relatively high, each flow contends with other flows to gain access to the wireless medium. Furthermore, we also have contention for the shared medium within a single flow, if a common channel is used for multiple hops.

Each flow created using HOVER initially goes over a non-optimal path, just as in the case of AODV-MR. However, upon expiry of the RREQ optimisation timer, the flow is switched to a better route comprising mostly of Mesh Routers and with optimal link

[2] Although, 802.11b can support only three orthogonal channels, we have configured the NS-2 802.11b physical layer to consider all channels to be orthogonal. This allows us to simulate the behaviour of radios that support a high number of orthogonal channels such as 802.11a.

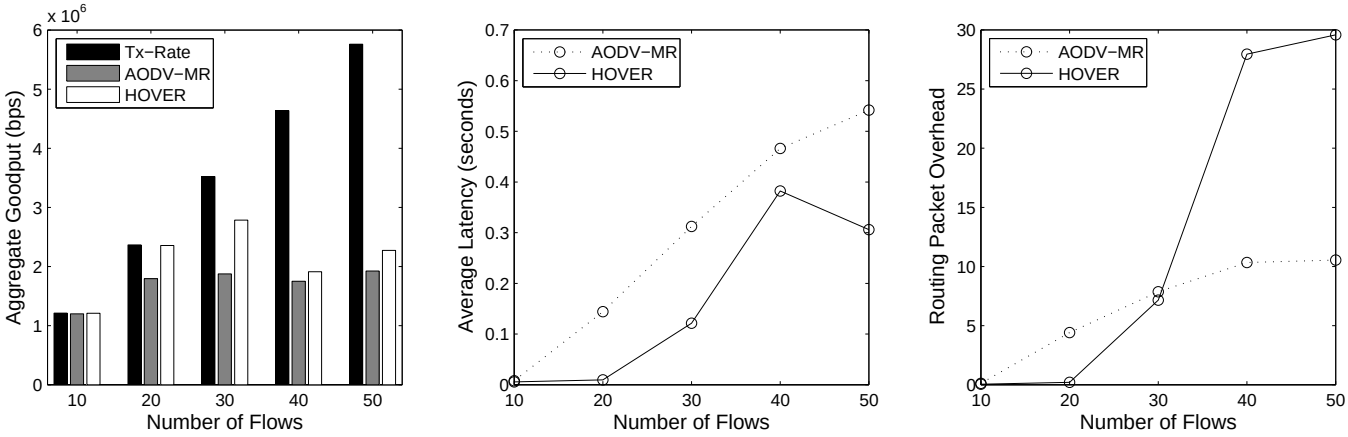


Figure 6: Test 2 Results - Varying the traffic load

selection. As a consequence, HOVER achieves a significantly higher goodput than AODV-MR.

The node-type aware routing mechanism allows HOVER to route most of the traffic via the Mesh Routers. Due to the availability of multiple links and the optimal link selection mechanism, HOVER is able to forward packets via paths with less contention and interference. As a result, the packet delays are significantly reduced.

The total number of control packets generated by HOVER is always higher than for AODV-MR, due to the fact that multiple copies of RREQs are forwarded. However, at zero Mesh Client speeds, HOVER achieves a lower routing overhead than AODV-MR, even though it generates a higher number of control packets. This is due to the fact that the routing overhead metric is computed as a ratio of control packets generated to successfully received data packets, which is higher in the case of HOVER.

Once the client speed is higher than zero, HOVER has a higher routing overhead. In this case routes are less stable and need to be re-established more often, and the overhead of sending multiple copies of RREQs and RREPs starts to become more evident.

4.3 Test 2 : Varying the Traffic Load

The results of Test 2 are depicted in Fig. 6. In this test, we increased the number of simultaneous 128 kbps flows from 10 to 50.

In this case, the aggregate application layer data rate is more than 5Mbps (40 x 128kbps). This essentially saturates the physical layer, which has a rough effective throughput of no more than 5 Mbps (Anderson & Youell 2002). However, HOVER achieves a significantly lower packet loss rate when the number of flows is less than 40.

The aggregate goodput achieved by HOVER remains higher than that of AODV-MR due to reduced packet losses at lower traffic loads. The packet delivery ratio for both protocols remains at almost 100% when the number of flows is set to 10. However, HOVER shows an improvement of around 25% in the packet delivery ratio over AODV-MR when the number of flows is set to 20 or 30.

The latency of the packets using HOVER remains more than 100 ms lower than that observed using AODV-MR, with 20 or more simultaneous data flows. The routing overhead of HOVER remains lower than that of AODV-MR when the number of flows remains below 40. This is essentially due to the low speed of the Mesh Clients (1 m/s) maintained in Test 2.

5 Prototype Implementation and Evaluation

5.1 Prototype Implementation

We implemented a prototype of HOVER based on AODV-UU (version 0.9.3)[3]. Some of the key changes to the code that were required to implement our extensions are summarised as follows:

- Contrary to claims, the current version of AODV-UU (0.9.3) does not correctly support multiple network interfaces per node. We made a number of changes to the original AODV-UU code to fix this problem.
- We added a mechanism to keep track of multiple links to neighbours and their respective quality metrics. This was achieved by extending AODV-MR's routing table.
- An optimisation timer was associated with each routing table entry. This timer is started when the first RREQ is received, and the optimal RREP is sent when the timer expires.
- AODV-MR's Hello mechanism has been modified to compute the link quality metric.

5.2 Testbed Set-up

To evaluate the efficacy of HOVER, we implemented a small testbed consisting of nine wireless nodes as shown in Fig. 7. The hardware and software setup are explained in the following sub-sections.

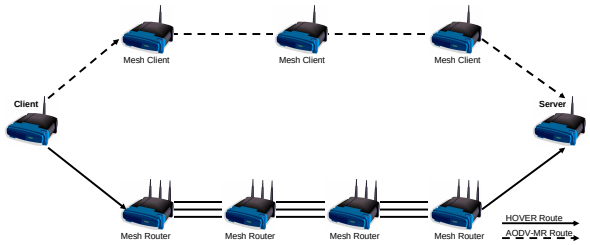


Figure 7: Hybrid Mesh Network Testbed

[3] http://core.it.uu.se/AdHoc/AodvUUImpl

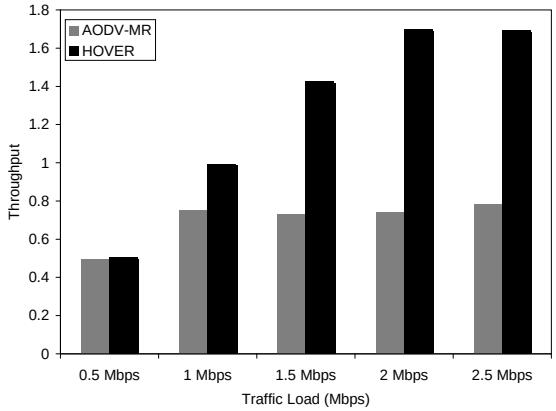


Figure 8: Throughput results under varying traffic loads

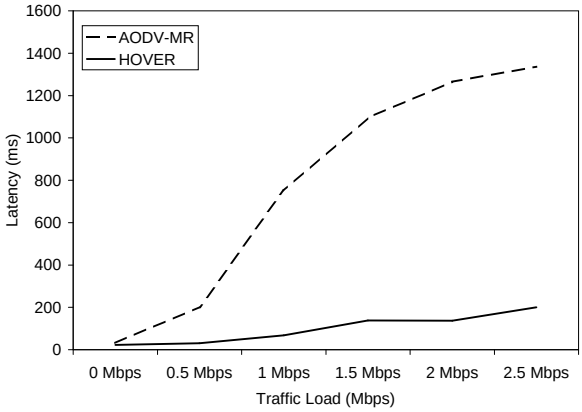


Figure 9: Latency results under varying traffic loads

5.2.1 Hardware

The testbed consists of four Mesh Routers shown at the bottom of Fig. 7. Mesh Routers are implemented as standard PCs (AMD Sempron 2800+ processor and 512MB of RAM), running Linux kernel version 2.6.8. Each Mesh Router is equipped with three mini-PCI CM9 (AR5212 802.11a/b/g) wireless radios, using the Madwifi driver SVN version r1639. One radio is tuned to 802.11b channel 1, and the remaining two are tuned to 802.11a channel 42 and 160 respectively. Linksys WRT54GL wireless routers were used as Mesh Clients and were running the openWRT Linux distribution. The WRT54GLs are equipped with one 802.11b radio only, tuned to 802.11b channel 1.

5.2.2 Software

We used Iperf[4] and Ping to measure the network throughput and latency respectively. Since all nodes are located within close proximity of each other in our lab, and therefore are within one-hop range of each other, we implemented virtual topology control via MAC layer filtering using iptables[5]. The resulting topology is illustrated in Fig. 7. Two possible paths exist between the client and the server: a four hop path via the Mesh Clients and a five hop path via the Mesh Routers.

5.3 Prototype Evaluation

A UDP session was established using Iperf between the server and the client, which are respectively positioned at the far right and left sides of Fig. 7. The offered load of each session was increased from 500 kbps to 2.5 Mbps. All results have been averaged over twenty individual test runs.

The results, shown in Fig. 8, indicate that AODV-MR always selects the first path, due to its smaller hop-count. HOVER also initially selects the first path, but after the expiry of the optimisation timer, switches over to the second (optimal) path. HOVER efficiently uses the multiple links that are available between nodes in the second path to increase the throughput significantly.

The latency results, shown in Fig 9, also confirm minimal contention for the physical wireless medium due to optimal channel selection. The results are in line with our simulation results, in which HOVER shows a notable improvement over AODV-MR in terms of latency. However, due to the absence of any inter-flow interference (Yang, Wang & Kravets 2005) in the testbed, the improvement in latency is notably higher in the testbed than in the simulation results.

[4] http://dast.nlanr.net/Projects/Iperf/
[5] http://www.netfilter.org

6 Conclusions

Hybrid Wireless Mesh Networks are composed of a combination of static Mesh Routers and mobile Mesh Clients. The Mesh Routers have significantly higher computation and communication resources compared with the Mesh Clients. However, current routing protocols do not discriminate between the two type of nodes and are, therefore, not able to exploit this heterogeneity. In this paper, we presented a number of extensions to the multi-radio AODV protocol that significantly improve its performance in terms of throughput and latency in Hybrid WMNs. Our extensions allow differentiation between node types and optimal use of the multiple links that are available at Mesh Routers. We implemented three methods to achieve this: node-type aware routing, link quality estimation, and optimal link selection. We have demonstrated the superior performance of HOVER over multi-radio AODV with the help of extensive simulations. We have also shown the practicality of our concept via an actual implementation, and provided realistic evaluations via testbed experiments.

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.

Anderson, J. K. & Youell, N. (2002), 'A Closer Look at WLAN Throughput and Performance', *Bechtel Telecommunications Technical Journal* **1**(1), 86–94.

Chakeres, I. & Belding-Royer, E. (2002), The utility of hello messages for determining link connectivity, *in* 'Proceedings of the 5th International Symposium on Wireless Personal Multimedia Communications', Vol. 2, pp. 504–508.

Chandra, R. & Bahl, P. (2004), MultiNet: Connecting to Multiple IEEE 802.11 Networks using a Single Wireless Card, *in* 'Proceedings of the Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)', Vol. 2, IEEE Press, pp. 882–893.

Couto, D. S. J., Aguayo, D., Bicket, J. & Morris, R. (2005), 'A high-throughput path metric for multi-hop wireless routing', *Wireless Networks* **11**(4), 419–434.

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.

Gupta, P. & Kumar, P. R. (2000), 'The Capacity of Wireless Networks', *IEEE Transactions on Information Theory* **46**(2), 388–404.

Johnson, D. B., Maltz, D. A. & Hu, Y. (2003), 'The Dynamic Source Routing Protocol for Mobile Ad hoc Networks (DSR)', *IETF MANET, Internet Draft* .

Ko, B. J., Padhye, J., Misra, V. & Rubenstein, D. (2005), Distributed Channel Assignment in Multi-radio 802.11 Mesh Networks. , Technical report, Columbia University.

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.

NS (1989), 'The Network Simulator', *http://www.isi.edu/nsnam/ns/* .

Perkins, C. E. & Bhagwat, P. (1994), Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers, *in* 'Proceedings of the SIGCOMM Conference on Communications, Architectures, Protocols and Applications', ACM Press, pp. 234–244.

Perkins, C. & Royer, E. M. (1999), Ad hoc On-Demand Distance Vector Routing, *in* 'Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications', pp. 90–100.

Perkins, C., Royer, E. M. & Das, S. (2003), 'Ad hoc On-Demand Distance Vector (AODV) Routing', *IETF RFC 3561* .

Pirzada, A. A. & McDonald, C. (2004), Establishing Trust in Pure Ad-hoc Networks, *in* 'Proceedings of the 27th Australasian Computer Science Conference (ACSC)', Vol. 26, Australian Computer Society, pp. 47–54.

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.

Pirzada, A. A., Portmann, M. & Indulska, J. (2006), Evaluation of MultiRadio Extensions to AODV for Wireless Mesh Networks, *in* 'Proceedings of the 4th ACM International Workshop on Mobility Management and Wireless Access (MobiWac)', pp. 45–51.

Pirzada, A. A., Portmann, M. & Indulska, J. (2007), Hybrid Mesh Ad-hoc On-demand Distance Vector Routing Protocol, *in* 'Proceedings of the Thirtieth Australasian Computer Science Conference (ACSC'07)', Vol. 29, pp. 49–58.

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, pp. 4–15.

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.

Yang, Y., Wang, J. & Kravets, R. (2005), Designing Routing Metrics for Mesh Networks, *in* 'Proceedings of the IEEE Workshop on Wireless Mesh Networks (WiMesh)', IEEE Press.

Product Flow Analysis in Distribution Networks With a Fixed Time Horizon

M. T. Wynn **C. J. Fidge** **A. H. M. ter Hofstede** **M. Dumas**

Faculty of Information Technology,
Queensland University of Technology, Australia.
Email: {m.wynn, c.fidge, a.terhofstede, m.dumas}@qut.edu.au

Abstract

The movement of items through a product distribution network is a complex dynamic process which depends not only on the network's static topology but also on a knowledge of how each node stores, handles and forwards items. Analysing this time-dependent behaviour would normally require a simulation algorithm which maintains a globally-synchronised system state. For a certain class of problem, however, where the simulation is required to stop in a consistent state but not necessarily maintain consistency at all times, we show that an algorithm that makes localised decisions only is sufficient. As a motivating example we consider the practical problem of product recalls, in which our primary concern is the state of the distribution network at a specific time after a batch of suspect items was released, but we do not necessarily care about intermediate states leading up to the final one.

Keywords: Network analysis; Product distribution.

1 Introduction

A product distribution network routes items from source nodes to their final destinations, with each item typically passing through several nodes en route. The route that each item takes is controlled by the logistic processes governing each node. Examples include mail systems, small-scale retail networks and large-scale shipping networks. Understanding the behaviour of distribution networks is vital to business process modelling, but is challenging because the dynamic properties of such a network cannot be predicted easily from its static topology. Instead, some form of simulation is usually required to analyse its behaviour.

Such an analysis would normally require us to use an algorithm which maintains a global notion of time throughout the simulated network. Unfortunately, maintaining global states is awkward when dealing with asynchronous distributed systems. Keeping the global state consistent at all points in the computation forces us to allow for the possibility that an action in one node completes mid-way through a concurrent action in another node. The resulting simulation algorithm thus needs to partition logically atomic steps, adding to its complexity. Also, the algorithm can be inefficient if the time-scale is fine-grained.

We have found, however, that for a certain class of distribution network problem there is no need to maintain a global state. Specifically, when the items moving through the network behave independently of one another, and when the desired result of the analysis is to discover the state of the network at some pre-determined time, we can use an algorithm that does not require synchronisation across the nodes. Instead, we define a global 'horizon' for the calculation and model each item's passage through the network separately until it reaches the horizon. The resulting calculation can be more efficient than the corresponding globally-synchronised simulation algorithm because each item can make progress in comparatively large time steps.

Here we use the highly-topical issue of product recalls as a motivating example to demonstrate the approach. Recent dangerous product recalls in Australia, including several triggered by extortion attempts, have proven enormously difficult and expensive for the companies involved (Safe 2005). Developing a reliable solution to this problem is highly significant: 'over-recalling' is needlessly expensive, whereas 'under-recalling' danger products can cost lives. Finding an efficient solution is also important, given the size of typical product distribution networks (Safe 2005).

Barcodes and RFID tags do not solve the problem of locating goods, because they are useful only after items have been located. Also, recent recalls have involved small, high-volume items, such as chocolate bars, paracetamol tablets, and baked goods (Industry Search 2006). Low-cost items such as these cannot be tracked cost-effectively using satellite technologies. Therefore, when a recall becomes necessary, we must rely on our knowledge of the product distribution network's typical behaviour in order to predict the likely location of suspect items. In this paper we present such an analysis algorithm based around the items themselves.

2 Related Work

Our interest here is an algorithm for analysing the behaviour of a physical distribution network represented as an acyclic directed graph with quantities attached to the nodes and arcs. There are numerous graph analysis algorithms for such capacitated digraphs, including algorithms for calculating shortest paths, minimum spanning trees, maximum capacity along a path, and so on (Berman & Paul 2005, Ch. 11). While relevant, none of these static analysis principles is directly applicable to our problem of identifying a particular state in the dynamic evolution

Acknowledgments We wish to thank the anonymous ACSC reviewers for their helpful comments. This research was funded by Australian Research Council grant DP0773012, *Rapidly Locating Items in Distribution Networks with Process-Driven Nodes.*

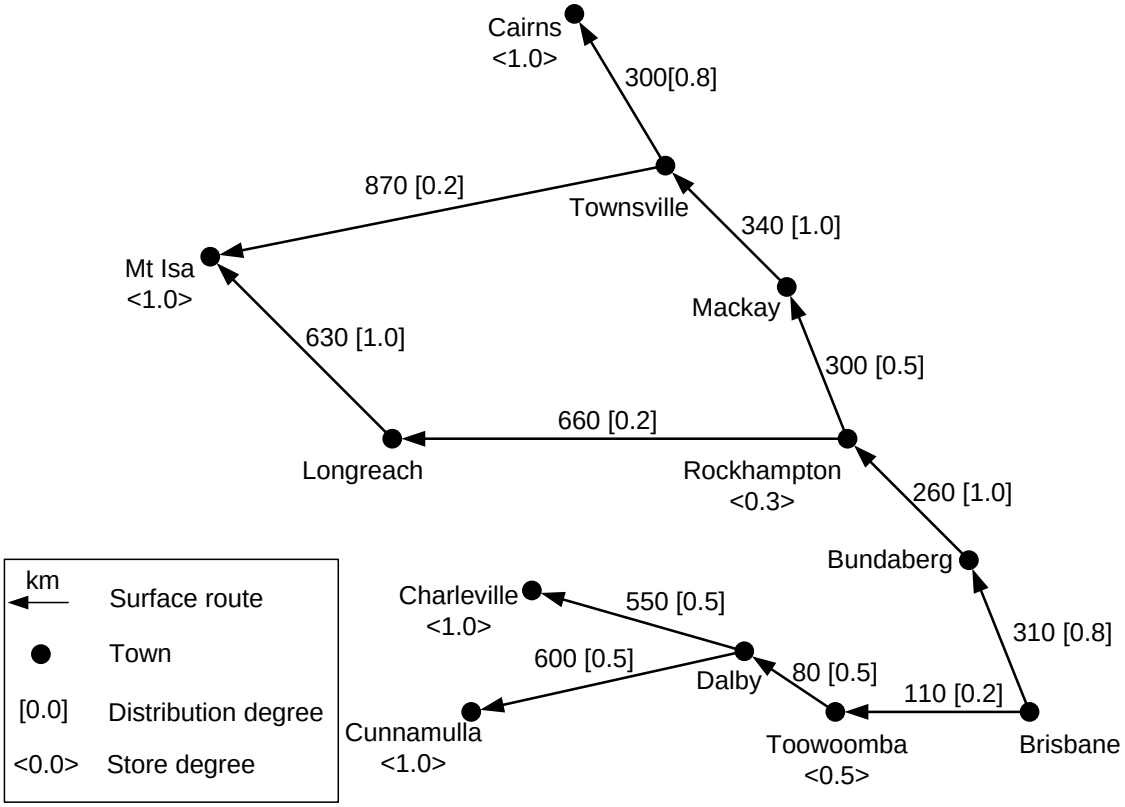


Figure 1: Major Queensland distribution routes from Brisbane

of a distribution network.

With respect to our motivating example of industrial product recalls, there are already several commercial products for managing product recalls in the manufacturing industry. However, these tools are primarily intended for products that are still within the confines of the factory or warehouse (Pulse Logistics Systems 2005, Hajnal et al. 2004), and are normally designed to help isolate the invoices and shipping orders for suspect products. They allow each item's first port of call to be identified, but it is expected that each of these distribution points will then be contacted individually in order to determine what has happened to the items in question (Seradex 2005). Our goal is instead to determine where suspect items are likely to be found after they have left the controlled facility and have started circulating in a physical distribution network.

There is extensive literature in the fields of logistics and supply chain management dealing with the analysis of physical distribution networks (Sarmiento & Nagi 1999). Techniques developed in this field support the design of such networks under various constraints and optimisation criteria like, for example, identifying optimal warehouse locations and vehicle routes (Daskin & Owen 2003). Other problems addressed in this field include performance measurement, such as estimating travel times between locations in a network (Lin et al. 2005). In contrast to this prior work, we do not deal with the design, evaluation or optimization of physical distribution networks. It is assumed that the network is given and that data on its topology, routing behaviour at each node and transportation delays are available. Given these data, we seek to estimate the possible location of products dispatched on a given date, within a fixed time horizon.

A problem closer to the one faced here is that of tracking information flow through electronic communication networks. Prior research on security-critical communications devices has shown how the potential destinations of classified data can be determined by treating circuitry schematics as digraphs and by modelling the routing of information through each node via the different operating 'modes' of the individual electronic components (Rae & Fidge 2005, Fidge & McComb 2006). A practical tool for information-flow analysis was developed from this theory (McComb & Wildman 2005). Although similar to the problem addressed in this paper, this previous work is distinct because, unlike physical items, 'information' is infinitely copiable. Therefore, this previous information flow algorithm traced all nodes through which information has passed, rather than identifying its 'current' location. Furthermore, this previous work did not incorporate the notion of elapsed time, or of a time horizon, both of which are central to this paper.

3 Motivating Example

As a motivating example we consider the topical problem of locating specific items circulating in large organisations or supply chain networks. This can be a security-critical or safety-critical issue when the item's status changes. For example, rapidly finding a memorandum circulating in a government department can be critical if the document's classification is changed from 'unclassified' to 'secret'. Tracking down a package that has been sent into a postal network is vital if authorities are told it may contain explosive materials. Similarly, locating frozen foods en route to retailers can be a public health issue if they come from a batch found to be contaminated with salmonella. In each of these cases the challenge is to isolate the likely location of particular items which have been released into a largely autonomous, or self-regulating, distribution network.

Consider the situation where a Brisbane-based company must issue an urgent recall for a product that was shipped 48 hours ago for distribution

throughout Queensland. Figure 1 shows all the company's known transport routes. With the geographical data expressed as a directed graph, it is possible to perform a transitive connectivity analysis to determine where the items of interest *may* have gone. However, this proves to be useless in practice. Assuming an average land speed of 60km/h, the items could have traveled 2,880km since they were released, making it possible to reach any node in the graph. In other words, our conclusion from this simple analysis is that the suspect items could be anywhere! This static analysis fails because it does not take into account how the items are routed at each node and precisely how long they take to travel between nodes.

Now assume that the company has more detailed information about the typical behaviour of each node in the distribution network, including how items are typically stored, handled and forwarded, including any seasonal differences. For example, the arcs in Figure 1 have been annotated with the distribution percentage between each pair of nodes, in square brackets. It shows, for instance, that 80% of all products arriving at Townsville are forwarded to Cairns and the remaining 20% are sent to Mt. Isa. Similarly, the storage percentage associated with a node, in angled brackets, represents the percentage of arriving products offloaded at a particular town for distribution locally. For instance, the figure shows that 50% of all products sent to Toowoomba are consumed in the local region. At sink nodes, i.e., the ends of the distribution chain, we assume that all products are offloaded. For instance, 100% of items arriving at Cairns are offloaded.

In practice this model can be developed and calibrated incrementally over time, using information supplied by distributors based on actual observations of products in the field. (However, if no information is yet available for a particular node a default worst-case model can be assumed. For the product recall problem, this is when the node forwards items on all its outgoing arcs and stores some locally, i.e., it distributes items as widely as possible.)

With this additional information, it is now possible to carry out a more detailed analysis for the possible locations of particular products. Our main concern when a product recall is ordered is to identify those towns at which dangerous goods amay be found. This means we are interested in the state of the network at a specific moment in time (usually the present moment) following the release of the items. We call this the 'horizon' of the analysis. However, we usually have no interest in how the network reached this state. These characteristics of the problem are central to our algorithm, since they allow us to perform calculations without the need to maintain consistency in the intermediate states leading up to the final one.

4 Basic Product Flow Algorithm

In this section, we describe our algorithm for performing product flow analyses up to a fixed time horizon. We first explain key concepts and the data structures central to the algorithm, and then the individual functions from which the algorithm is composed. Finally, we give a step-by-step explanation of the algorithm's calculation, using the motivating example from Section 3 above.

4.1 Data Structures

In this section the data structures and access functions which are used by the algorithm are defined.

4.1.1 Network Topology

To perform the analysis, it is critical that the information about distribution routes and connections between different towns is known in advance. A distribution network can be considered as a 'directed graph' with vertices representing distribution nodes and edges representing the connections between two nodes. The role of a node in a particular distribution route can vary. A node could represent an initial source of the distribution route (e.g., Brisbane in our sample network), or a final sink node for products (e.g., Cairns in our example), or it could be an intermediate destination for a product, acting as both a target and a source (e.g., Rockhampton). We also assume that the graph of a distribution network is acyclic.

To formalise this notation, let type *Node* represent the distribution nodes in the network graph. A possible connection between two nodes (a source and a target) is defined as an element of relation $Connection \subseteq Node \times Node$. Function $src : Connection \rightarrow Node$ returns the source node of a connection and function $tgt : Connection \rightarrow Node$ returns the target node of a connection.

4.1.2 Distribution Functions

The more detailed information we have about how each node operates, the more reliable the results of the analysis will be. For our purpose, we need to know how a node distributes products to its outgoing connections and what proportion of products are offloaded locally, if any. We use the term 'distribution degree' to represent the percentage of products distributed to each connection from a node and 'store degree' to represent the percentage of products offloaded at a node.

The distribution degree for a connection is defined as real-valued function $degree : Connection \rightarrow [0 \ldots 1]$. The store degree for a node is defined as partial function $storeDegree : Node \nrightarrow [0 \ldots 1]$, since not all nodes will store items locally. The set of nodes where items could be offloaded is the domain of this function, i.e., *dom storeDegree*. Let *Set* be the powerset type constructor, N be the set of natural numbers, and type $NodeStorage = Node \times N$ comprise node-number pairs. Then function $split : Set(Connection) \times N \rightarrow SetNodeStorage$ makes use of the *degree* and *storeDegree* functions to return a set of node-number pairs that represents the number of items to be sent to a given node. This function ensures that the total number of products forwarded to various nodes is the same as the total number received.

4.1.3 Packages

We use the term 'package' to describe a parcel of products that is being distributed on a network route. We assume that a package contains a number of items of a given product. It is possible to split a package into two or more non-empty packages, preserving the total number of items, that can be then distributed further in the network. The route that a package follows is solely dependent on the logistical behaviour of the network nodes (i.e., packages are not individually addressed).

In the algorithm we keep track of the number of items in each package, the time the package arrived at its current (or last) node, the location of the package (the source and target of the last distribution step) and whether or not a package has reached its final destination. Let *Package* represent the type of a package that is routed across a network. The

Function `DistributePackage`

Input: $p : Package$
Output: $P' : SetPackage$
Pre: $\neg finalDestination(p)$

begin
  $SetConnection\ NextConnections := \{c : Connection | tgt(location(p)) = src(c)\}$;
  $SetNodeStorage\ SplitCount := split(NextConnections, count(p))$;
  for $c \in NextConnections$ **do**
    $Package\ p' := new\ Package()$;
    $timeStamp(p') := timeStamp(p) + delay(c, p)$;
    $location(p') := c$;
    $count(p') := SplitCount(tgt(c))$;
    $finalDestination(p') := false$;
    $P' := P' \cup \{p'\}$;
  end
  if $tgt(location(p)) \in dom\ storeDegree$ **then**
    $Package\ p' := new\ Package()$;
    $timeStamp(p') := timeStamp(p)$;
    $location(p') := location(p)$;
    $count(p') := SplitCount(tgt(location(p)))$;
    $finalDestination(p') := true$;
    $P' := P' \cup \{p'\}$;
  end
  return P';
end

functions associated with a package are as follows: $count : Package \rightarrow N$ returns the number of items in a package, $timeStamp : Package \rightarrow N$ returns the absolute time delay (in hours) since a package started its journey, $location : Package \rightarrow Connection$ returns the current location of a package, which is interpreted to mean that the package is either en route on this connection or has reached the connection's target node, and $finalDestination : Package \rightarrow Boolean$ returns whether or not the package has reached its final destination.

The propagation delay, in hours, of a package traversing a connection is determined by function $delay : Connection \times Package \rightarrow N$. For now, we treat this function as a simple mapping from connections to fixed delays. In Section 5, we demonstrate how this function can be made more realistic by introducing the notions of package handling delays, transport delays, and lead time delays.

4.1.4 Time Horizons

The time horizon indicates when the analysis should terminate and it is an important feature of our analysis technique. The intention is to calculate the way packages move through the network until a predetermined time is reached. At this time it is possible that all packages have arrived at their final destinations and the network is quiescent. Alternatively, only some of the packages may have reached their final destinations and others may be still en route. This latter situation is the challenge of particular interest to us. In effect, we want the calculation to stop in a state which gives a consistent snapshot of where each package is in the network when the time horizon is reached. The horizon itself is provided as an input to the algorithm and it is represented as an integer denoting elapsed hours.

4.1.5 Network State

The algorithm's state represents the status of packages at intermediate steps during their journey as well as the status of the various packages at the final state, when the analysis reaches a given time horizon. The state is made up of three sets of packages, as explained below.

The analysis stops when the predetermined time horizon is reached by all packages. When completed the calculation returns the following three sets of packages:

- Final: This set represents all the packages that have reached their final destinations before or at the given time horizon. If all packages have reached their final destinations, the other two sets will be empty.
- Leading edges: This set represents all the packages that are in transit, i.e., traversing a connection, when the time horizon is reached. It includes the (future) times at which they will reach their next node.
- Trailing edges: This set also represents the packages that are in transit when the time horizon is reached. However the additional attribute in this set is the (past) time at which each package left its previous node.

To support this, the algorithm's data structure *State* is declared as $SetPackage \times SetPackage \times SetPackage$, in which function *final* returns the first set of packages, function *leading* returns the second set of packages, and function *trailing* returns the third set of packages in a state.

4.2 Baseline Algorithm

Here we describe the three functions that make up the basic algorithm for product flow analysis.

- *DistributePackage*: This function describes how a package could be split into child packages and distributed to the connected nodes in one step using the distribution functions available for each node. Function *DistributePackage* (above) takes a package as input and returns a set of packages that are to be distributed in the next step.
- *ProgressPackages*: This function contains the program logic whereby the timestamp of a package is compared with the given time horizon to

Function `ProgressPackages`

Input: $packages : SetPackage, horizon : N, sn : State$
Output: $state : State$

```
begin
    state := sn;
    for p ∈ packages do
        for p' ∈ DistributePackage(p) do
            if timeStamp(p') ≤ horizon ∨ finalDestination(p') then
                final(state) := final(state) ∪ {p'};
            end
            else
                leading(state) := leading(state) ∪ {p'};
                Package p_t := new Package();
                timeStamp(p_t) := timeStamp(p);
                location(p_t) := location(p);
                count(p_t) := count(p');
                finalDestination(p_t) := finalDestination(p');
                trailing(state) := trailing(state) ∪ {p_t};
            end
        end
    end
    return state;
end
```

Function `PerformFlowAnalysis`

Input: $I : SetPackage, horizon : N$
Output: $state : State$

```
begin
    State state := new State();
    SetPackage P := I;
    state := ProgressPackages(I, horizon, state);
    while P ≠ final(state) do
        P := final(state);
        final(state) := ∅;
        state := ProgressPackages(P, horizon, state);
    end
    return state;
end
```

determine how the state of the analysis should be updated with the information about the child packages. If the timestamp of a child package is less than or equal to the horizon, the package is put back into the final set for further distribution and/or storage. Otherwise, the package is stored in the leading and trailing sets. This indicates that the package is in transit and it has left the source at the time recorded in the trailing set and it should arrive at the target node at the time recorded in the leading set. Function *ProgressPackages* (above) takes as input a set of packages, produces a number of children packages for distribution (in one step) and then returns the results based on the given horizon. This function makes use of the *DistributePackage* function.

- *PerformFlowAnalysis*: This is the main function that calls the *ProgressPackages* function iteratively until the final state is reached for a given horizon. Function *PerformFlowAnalysis* (above) performs a fixed-point calculation and stops when there is no change in the final set of packages in the state. The function takes as inputs the initial set of packages to distribute and the time horizon to stop the analysis.

To validate this algorithm, we have developed a prototype implementation (downloadable from `http://www.yawlfoundation.org/research/simulation.php`).

4.3 Illustration

Here we illustrate the functioning of the algorithm using our earlier motivating example (Section 3). Consider the scenario where a Brisbane-based company must issue an urgent recall for a product which was shipped as a package containing 1000 items 48 hours ago for distribution as per the network shown in Figure 1.

Using the data given in Figure 1, regarding the distances between nodes and the way items are routed, we calculated where the products could be after 48 hours. Figure 2 shows the 'final state' information from the analysis overlayed with the network graph. The algorithm starts with a package of 1000 items, with timestamp '0' at Brisbane and horizon of '48'. The sample distribution degrees and store degrees for each node are then used to determine the appropriate splitting of packages. For instance, from Brisbane a package with 800 items (due to the 0.8 degree assigned to the connection Brisbane-Bundaberg) is sent to Bundaberg and a package with 200 items is sent to Toowoomba (due to the 0.2 degree assigned to the connection Brisbane-Toowoomba). The package arrives at Toowoomba after a certain delay, in this case 7 hours from the time it left Brisbane. The timestamp represents the arrival time of the package at a node and it is represented using the "@" symbol. At Toowoomba, a package with 100 items is stored (due to the 0.5 store degree assigned to the node Toowoomba) and a package with 100 items is

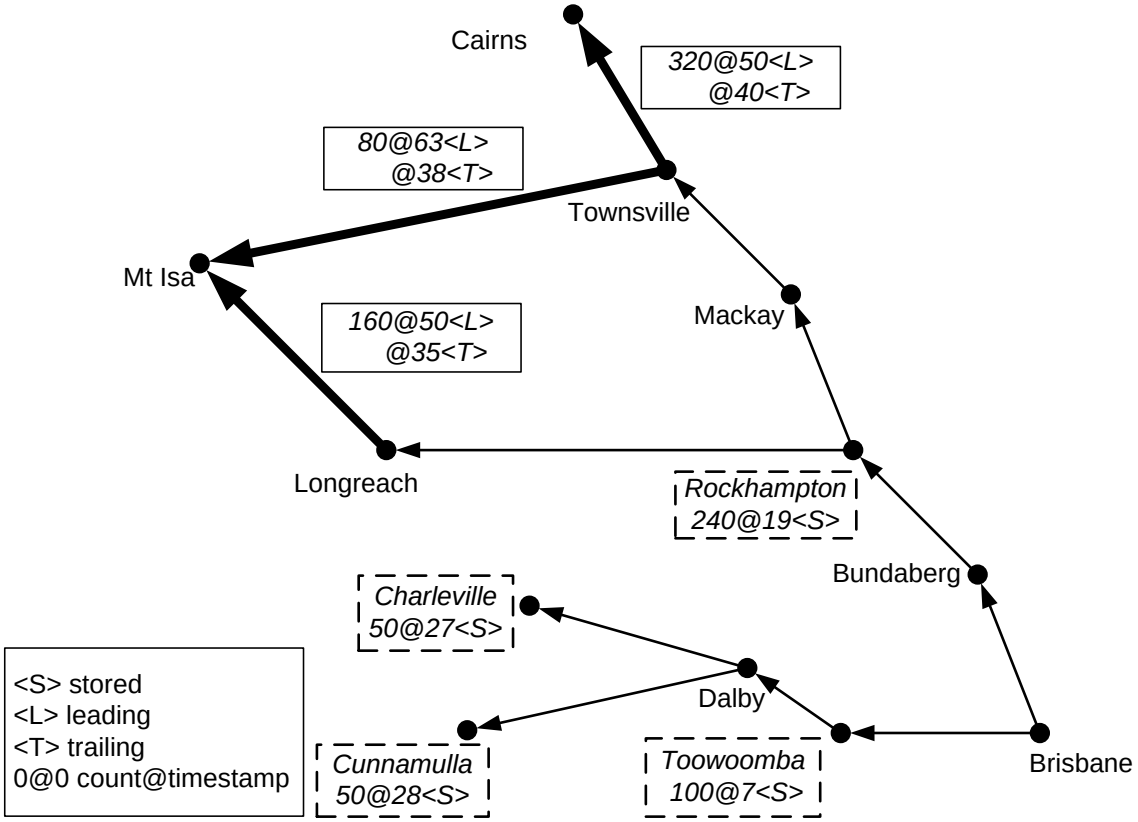


Figure 2: Possible locations of 1000 items distributed from Brisbane 48 hours ago

forwarded to Dalby. From Dalby, the packages are distributed to Charleville and Cunnamulla as shown in the figure. As the example network shows Cunnamulla and Charleville as final destinations, all items are offloaded at these destinations and marked as 'stored' (dashed boxes in the diagram). For other packages, the distribution continues until the given horizon is reached. From the figure, one can see that three packages are still en route after 48 hours has elapsed (solid boxes in the diagram with bold arrows for connections). These packages are shown with the trailing and leading times. For instance, there is a package containing 160 items travelling between Longreach and Mt. Isa. The package is expected to arrive at Mt. Isa at time 50 and it left Longreach at time 35. Similarly, there is another package containing 80 items due to arrive at Mt. Isa at time 63 from Townsville. This snapshot information is very useful to determine not only where items could be located but also to eliminate locations where items could not be.

5 Computing Delays

The basic algorithm uses a simple delay function to determine how long it takes a package to get from one node to the next. In this section, we show how to compute a more realistic delay duration by taking into account the transport delay based on the distance between connections, the transport mode and its speed, the handling delay at each node based on the availability of resources, and the lead time delay based on the operating hours of a distribution node.

To do this the delay function is extended to take into account the transport delay, the handling delay, and the lead time delay as follows.

Function `delay`

Input: $c : Connection, p : Package$
Output: $delay : N$
begin
  $Node\ n := tgt(location(p))$;
  $Day\ d := day(arrivalDayTime(p))$;
  $Time\ t := time(arrivalDayTime(p))$;
  $delay := transportDelay(c) +$
    $handlingDelay(n, count(p)) +$
    $leadtimeDelay(n, d, t)$;
  return $delay$;
end

We first describe how the transport delay can be calculated based on the various modes of travel and corresponding speed, and how the handling delay can take into account the employee's work rates at each node. (Of course, further data about transport speeds and handling rates could be incorporated into the algorithm in practice.) To do this we assume that the following additional functions are available for a connection: $distance : Connection \to N$ and $speed : Connection \times Mode \nrightarrow N$, where type $Mode = \{air, surface\}$. The transport delay can be then calculated as follows:

$$delay := (integer)distance(c)/speed(c, mode(c)) .$$

Further, assume that the following additional functions are available for a node: $empNum : Node \to N$ which returns the number of employees and $rate : Node \to N$ which returns the average number of items that could be handled by an employee per hour. From this information, the handling delay can be calculated as follows:

$$delay := (integer)items/rate(n) * empNum(n) .$$

We can also introduce a realistic lead time delay function either based on a weekly schedule or calen-

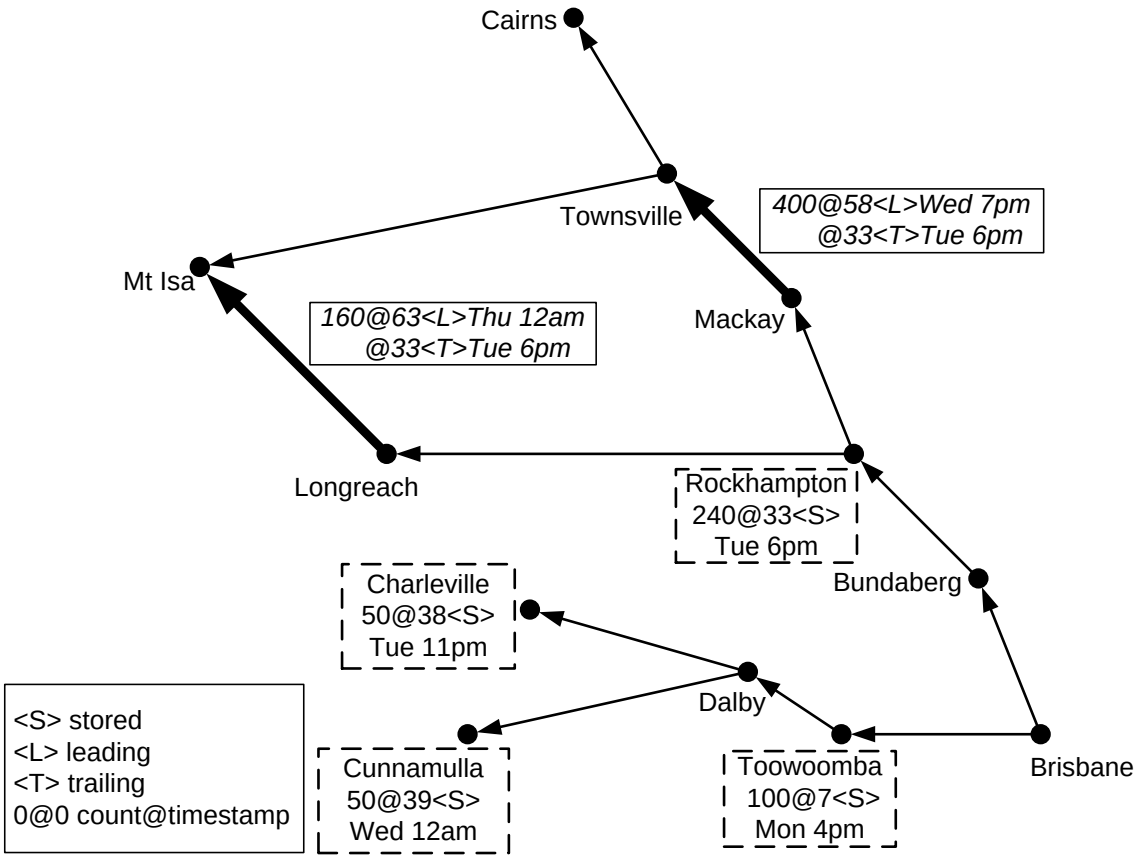


Figure 3: Locations of 1000 items 48 hours after being distributed from Brisbane on Monday at 9am

dar time. We base the weekly schedule on the known operating hours of each distribution node. For instance, a distribution centre may be open between 9–5pm Monday to Friday and 10–4pm on Saturdays but is closed on Sundays. In addition, we assume that the timestamps associated with packages are based on calendar, rather than elapsed, time. For instance, we may know that a package was sent on a Monday morning at 9am.

We can then determine where items from that package are in 48 hours time by taking into account the operating hours of the distribution nodes along the way. In this situation the location of the items could vary significantly due to effect of distribution nodes being either open or closed when a package arrives.

Let data type *Day* represent days of the week [Mon...Sun], and *Time* represent hours of the day [00...23]. Constructor $DayTime : Day \times Time$ returns the combination of the two, and selectors $day : DayTime \rightarrow Day$ and $time : DayTime \rightarrow Time$ extract the component parts of a timestamp. Let function $next : Day \rightarrow Day$ return the next day of the week, and function $addHours : DayTime \times N \rightarrow DayTime$ return the new day and time after adding the number of hours given as N, both using circular arithmetic.

Also let functions $SB : Node \times Day \nrightarrow Time$ and $CB : Node \times Day \nrightarrow Time$ return the start of business and close of business times of a distribution node, respectively. Function $DayOff : Node \times Day \nrightarrow Boolean$ returns whether a distribution node is closed on a certain day. New function $arrivalDayTime : Package \rightarrow DayTime$ returns the arrival day and time of a package.

We can now define function *leadtimeDelay* to take as inputs a distribution node and a day and a time (which represent the arrival day time of a package) and return the delay in hours. The resulting delay is calculated based on the arrival time of the package, the number of days that the distribution node is closed and the opening hours of the next business day.

Function `leadtimeDelay`

Input: $n : Node, d : Day, t : Time$
Output: $delay : N$
begin
  $delay := 0$;
  if $t \geq CB(n, d)$ **then**
    $delay := 24 - t$;
    while $DayOff(n, next(d))$ **do**
      $delay := delay + 24$;
      $d := next(d)$;
    end
    $delay := delay + SB(n, d)$;
  end
  return $delay$;
end

This delay calculation, including a weekly calendar, has been implemented in the prototype analysis program. In the calculations, we assume that it is possible for a package to arrive at any point in time (even during the closing hours). However, package are only forwarded to their next destination during opening hours.

As an example, Figures 3 and 4 provide a comparison of two calculations involving packages of the same size (1000 items) and the same fixed time horizon (48 hours). The difference between them is the day of the week on which the package is assumed to have started its journey. From Figure 3 it can be seen that if the package is sent on Monday at 9am, some of the items will reach their final destinations of Charleville and Cunnamulla before the expiry of the 48 hours time horizon. Other items will be in transit

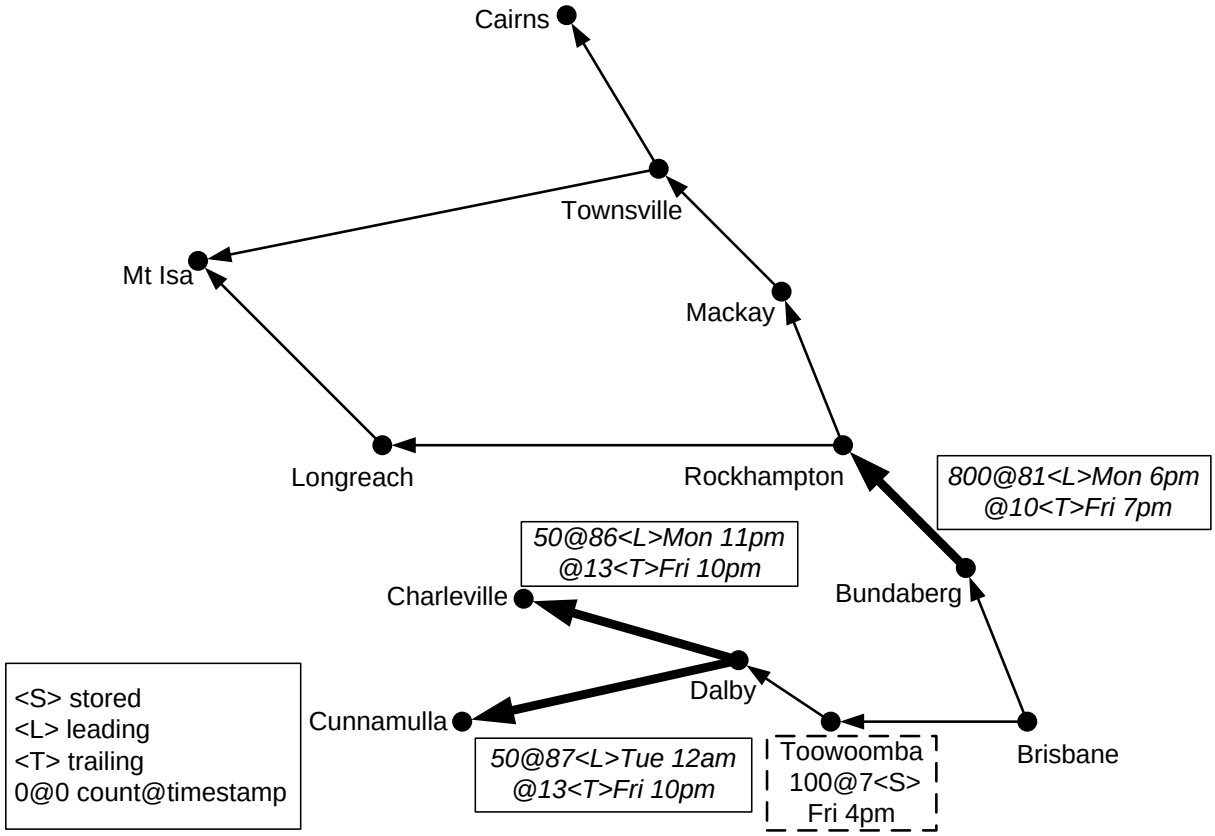


Figure 4: Locations of 1000 items 48 hours after being distributed from Brisbane on Friday at 9am

beyond Longreach and Mackay. However, Figure 4 reveals a quite different situation if the package is sent on Friday morning. Thanks to the processing delay introduced by the weekend, no items will have reached Charleville or Cunnamulla when the time horizon is reached, nor will any have gone as far north as Rockhampton. Being aware of the dramatic difference between these two outcomes would have a major impact on the speed with which a recall of dangerous goods could be conducted.

Another alternative extension to the basic algorithm is to take into account the actual date that a package has been sent, e.g., 24/12/2007, rather than the weekday. Function *leadtimeDelayDate* (below) describes how the delay can be calculated using a calendar date.

Function `leadtimeDelayDate`

Input: $n : Node, d : Date, t : Time$
Output: $delay : N$
begin
  $delay := 0$;
  if $t \geq CBDate(n, d)$ **then**
    $delay := 24 - t$;
    while $Holiday(n, nextDate(d))$ **do**
      $delay := delay + 24$;
      $d := nextDate(d)$;
    end
    $delay := delay + SBDate(n, d)$;
  end
  return $delay$;
end

Here data type $Date$ is $Day \times Month \times Year$ representing a date such that Day is in range $[0 \ldots 31]$, $Month$ is in $[0 \ldots 12]$ and $Year$ is a 4-digit number. Let function $nextDate : Date \rightarrow Date$ return the next day. Similar to the functions defined for $DayTime$, we can now define a set of functions for calendar dates that take into account the opening and closing times for a particular date as well as public holidays. To do this we would let partial functions $SBDate : Node \times Date \nrightarrow Time$ and $CBDate : Node \times Date \nrightarrow Time$ return the opening and closing times respectively of a distribution node; partial function $Holiday : Node \times Day \nrightarrow Boolean$ return whether or not a distribution node is closed on a certain date; and define functions $arrivalDate : Package \rightarrow Date$ and $arrivalTime : Package \rightarrow Time$.

6 Conclusion

Simulating the behaviour of a communications or transport network is normally a complex and challenging computation, requiring maintenance of a consistent global state across all nodes. However, we have identified an algorithm for analysing such behaviours that relies on local states only, provided that objects move through the network independently, and our overall goal is to determine their location at a predetermined time. The algorithm determines the way each item, or package of items, moves through the network, but achieves a consistent global state only when all items reach the fixed time horizon.

The usefulness of the algorithm was illustrated by showing how it can be applied to the problem of product recalls in distribution networks. It was also shown how the basic algorithm could be made more realistic by, for instance, incorporating a notion of calendar, rather than just elapsed, time.

Finally, we note that the algorithm described in this paper can be extended further in a number of ways:

- Rather than showing how a batch of items is split-up and distributed throughout the network, the probability that a particular single item arrives at different destinations can be calculated merely by changing the way we interpret the 'dis-

tribution degree'. Instead of representing a percentage split of items, it can be viewed as a percentage probability that the item of interest follows a particular route. The number at each node when the calculation ends then denotes the probability that the item is there at the time horizon.

- The functions for accounting for the handling and transport times could be made more realistic by changing them from simple numbers into probability distributions, based on observed or measured data. (In this case we would want to associate confidence intervals with the items' locations.)

A more dramatic extension would be to relax the assumption that items do not interact with one another. Doing this would require basing computational steps around nodes rather than packages. Nevertheless, it may be useful to introduce this concept to allow, for instance, modelling of the situation where a delivery truck will not depart from a node until enough items have arrived to fill it.

References

Berman, K. A. & Paul, J. L. (2005), *Algorithms: Sequential, Parallel and Distributed*, Thomson. ISBN 0-534-42057-5.

Daskin, M. & Owen, S. (2003), Location models in transportation, *in* R. Hall, ed., 'Handbook of Transportation Science', Kluwer, pp. 321–370.

Fidge, C. J. & McComb, T. (2006), Tracing secure information flow through mode changes., *in* V. Estivill-Castro & G. Dobbie, eds, 'Computer Science 2006, Twenty-Ninth Australasian Computer Science Conference (ACSC2006)', Vol. 48 of *Conferences on Research and Practice in Information Technology*, Australian Computer Society, pp. 303–310.

Hajnal, E., Kollar, G. & Lang-Lazi, M. (2004), 'IT support and statistics in traceability and product recall at food logistics providers', *Periodica Polytechnica Chemical Engineering* **48**(1), 21–29.

Industry Search (2006), 'Forty lines of Top Taste products are on recall nationwide', `http://www.industrysearch.com.au/news/printarticle.asp?id=20402`. Accessed June 2006.

Lin, H.-E., Zito, R. & Taylor, M. (2005), 'A review of travel-time prediction in transport and logistics', *Proceedings of the Eastern Asia Society for Transportation Studies* **5**, 1433–1448.

McComb, T. & Wildman, L. (2005), SIFA: A tool for evaluation of high-grade security devices., *in* C. Boyd & J. M. G. Nieto, eds, 'Information Security and Privacy, 10th Australasian Conference (ACISP 2005)', Vol. 3574 of *Lecture Notes in Computer Science*, Springer, pp. 230–241.

Pulse Logistics Systems (2005), 'Product recall software added', `http://www.pulse.com.au/news_detail.asp?NewsID=55&Source=News`. Accessed January 2005.

Rae, A. & Fidge, C. (2005), 'Information flow analysis for fail-secure devices', *The Computer Journal* **48**(1), 17–26.

Safe, M. (2005), 'Mars attack', *The Weekend Australian Magazine, 10–11 September* pp. 18–23.

Sarmiento, A.-M. & Nagi, R. (1999), 'A review of integrated analysis of production-distribution systems', *IIE Transactions* **31**(11), 1061–1074.

Seradex (2005), 'Product recall—Seradex ERP module', `http://www.seradex.com/Product_Recall_Module.shtml`. Accessed January 2005.

Experiments in the Dynamics of Phase Coupled Oscillators When Applied to Graph Colouring

Sofianto Lee and Raymond Lister
Faculty of Information Technology
University of Technology Sydney
{sofianto,raymond}@it.uts.edu.au

Abstract

This paper examines the capacity of networks of phase coupled oscillators to coordinate activity in a parallel, distributed fashion. To benchmark these networks of oscillators, we present empirical results from a study of the capacity of such networks to colour graphs. We generalise the update equation of Aihara et al. (2006) to an equation that can be applied to graphs requiring multiple colours. We find that our simple multi-phase model can colour some types of graphs, especially complete graphs and complete k-partite graphs with equal or a near equal number of vertices in each partition. A surprising empirical result is that the effectiveness of the approach appears to be more dependent upon the topology of the graph than the size of the graph.

Keywords: graph colouring, phase coupled oscillators.

1 Introduction

Observations of the phenomena of coupled oscillators date back to the early seventeenth century, when Christiaan Huygens noticed that the pendula of two of his clocks, suspended side-by-side, always settled into swinging in opposite directions, even after he disturbed the position of the pendula (Bennett *et al*., 2002; Strogatz, 2003). In 1680, Engelbert Kaempfer reported another form of phased coupled oscillation, in the synchronous flashing of hundreds of fireflies on trees along the Chao Phraya River in Thailand (Buck & Buck, 1976). Many similar instances of naturally occurring synchronization have since been discovered, such as in heart pacemaker cells and in neural networks (Camazine *et al*., 2001).

Fireflies generate light from the lantern in the abdomen; it usually takes about 800 milliseconds to recharge the lantern and 200 milliseconds to produce a spark; the process may then repeat. Formal models of this behaviour describe a single firefly as an oscillator with a phase $0 \leq \theta \leq 2\pi$ and period ώ. For a large proportion of each cycle, the oscillator is recharging and therefore discharging is impossible. For the remaining portion of the cycle, the firefly/oscillator is ready to discharge or "fire". If the firefly/oscillator is operating in isolation from other firefly/oscillators, then it fires at $\theta = 2\pi$. If a firefly/oscillator is not operating in isolation, has completed recharging, and sees sufficient light (stimulus) from neighbouring fireflies, the firefly/oscillator can adjust its phase slightly so as to bring itself closer to synchronization with the other firefly/oscillators (Camazine et al., 2001). Mirollo & Strogatz (1990) demonstrated, by mathematical proof and computer simulation, the conditions under which a fully connected network of oscillators will synchronise.

Networks of oscillators have properties that make them an interesting approach to coordinating activity in large networks of simple computational elements. First, the synchronization mechanism of the oscillators is parallel and distributed – no global coordination is required. Second, the oscillators can be implemented in hardware with very simple circuitry, making it a promising approach for massive networks of tiny processing elements. In fact, the approach has already received some attention for synchronization in ad-hoc sensor networks (Hong & Scaglione, 2003; Lucarelli & Wang, 2004; Werner-Allen et al., 2005), and the coordination of multi agent systems (Bettstetter, 2006; Spong, 2006).

1.1 Anti-phase Synchronisation with Two Oscillators

Phase coupling need not be confined to phase synchronisation (i.e. where the phase difference of oscillators is 0). In some applications, the desired effect may be to have the computational elements differentiate into two or more groups. One of the simplest models of anti-phase synchronisation was studied by Aihara et al. (2006). They studied the mating calls of rain frogs, which they modelled as a network of exactly two oscillators, where the oscillators were intended to interact in such a way that they would settle into having a phase difference of π.

In the Aihara *et al*. model, the two frogs/oscillators are denoted *a* and *b*. The phases of the frogs/oscillators are denoted θ_a, θ_b with the respective frogs calling when their phase is zero, and the frequency of the oscillators are denoted ω_a, ω_b. The dynamic of oscillator *a* in isolation from oscillator *b* is described by:

$$\frac{d(\theta_a)}{dt} = \omega_a \quad \ldots (1)$$

By extending Ermentrout & Rinzel (1984)'s model of firefly entrainment, Aihara *et al.* specified the interaction between the two oscillators as follows:

$$\frac{d(\theta_a)}{dt} = \omega_a - g(\theta_b - \theta_a - \alpha)$$
$$\frac{d(\theta_b)}{dt} = \omega_b - g(\theta_a - \theta_b - \beta) \quad \ldots (2)$$

Where g is a 2π periodic function, and the constants α and β are frustration parameters (we will assume α = β). Typically g is $K\sin(\theta_a - \theta_b - \beta)$ where the constant K is the coupling strength

Aihara *et al.* were able to show that a stable equilibrium phase difference $\varphi = \theta_a - \theta_b$ of π exists between the two oscillators provided $|\omega_a - \omega_b| << K$.

We performed a computer simulation of the Aihara *et al.* model. The phases of the two oscillators were randomly initialised and we used a coupling strength K of *0.1*. The waveforms in Figures 1 and 2 show the phases of the two oscillators, where they are initially out of phase by approximately 0.4 of a radian, finally reaching a stable anti-phase difference of ~3.14 from the 14th cycle onwards.

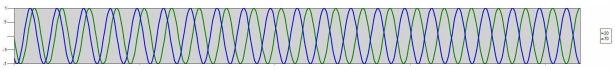


Figure 1: Evolution of the phases of two oscillators, which are eventually out of phase by π.

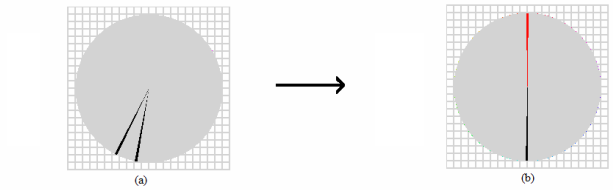


Figure 2: An alternative visualisation of the simulation from Figure 1. The wheel on the left shows the initial phase of each oscillator, which are similar. The wheel on the right shows the simulation at a later stage, when the phases of the two oscillators are separated by π.

1.2 Graph Colouring

In this paper, we further investigate the computational power of networks of oscillators. Like Hopfield & Tank (1985, p. 142), we believe that the computational power of such networks is best characterized by studying the behaviour of such networks when applied to difficult but well understood combinatorial optimization problems. Consequently, we have chosen to study the dynamics of networks of oscillators when applied to graph colouring.

The task of colouring a graph involves an assignment of colours to vertices in the graph such that no two vertices that share an edge have the same colour (Garey & Johnson, 1979). In our models, each node of the graph is an oscillator. Two oscillators are coupled if the respective nodes in the graph are connected. The colour of a node is represented by the phase of the oscillator. To visualise the graph colourings, we use a colour scheme that maps the phase of the oscillator in a 2π periodic system to a colour in the RGB (Red, Green, Blue) domain.

Wu (2002) conducted some simple experiments using oscillators to perform graph colouring. He conducted computer simulations for 300 graphs with the number of vertices ranging from 4 to 16, and where all graphs were known to be 2- or 3-colourable. His system coloured all 2-colourable graphs correctly, with a single exception, and also coloured approximately 80% of the 3-colourable graphs correctly. However, there are several limitations in Wu's study:

- Wu's approach to graph colouring was hybrid, where an initial colouring from the oscillators was subsequently "cleaned up" by an algorithm. Since our interest is in using oscillators to coordinate real networks, a hybrid approach is not practical: we need a purely parallel, distributed algorithm.
- Wu did not consider problems where more than three colours are required.
- Wu did not consider the effect of the graph topology on the effectiveness of the network of oscillators.
- Wu only considered the final state of his system, not the dynamics leading to the final state.

In this paper, we address these limitations in Wu's study. Furthermore, we generalise the Aihara *et al.* model so that it can be applied to more general graph colouring problems.

2 Two-Colouring in a Plane: The Ising Model

As a preliminary experiment, we chose to apply the Aihara *et al.* model to a simple and very well understood 2-colouring problem, the two-dimensional Ising Spin Problem (Kindermann & Snell, 1980). In this problem, which is illustrated in Figure 3, the nodes of the graph can be thought of as squares in a plane. Two nodes of the graph are adjacent if the corresponding squares share a common edge; therefore, each node is connected to four other nodes. It is obvious that such a graph can be 2-coloured, as shown in Figure 3.

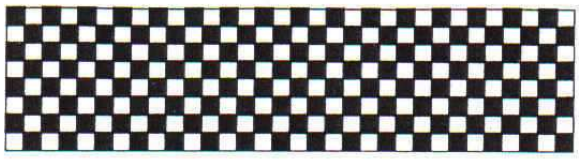


Figure 3: An example of the Ising Spin problem.

The Ising Spin problem is an interesting benchmark for two reasons. First, a planar mesh of computational elements is a realistic model of how a network of oscillators may be organised. Second, while the optimal

solution is obvious, simple distributed algorithms – where each computational element can only "see" its four neighbours – do not reliably produce the optimal 2-colouring. For example, Lister (1992) showed that, for an 8×32 problem, like that shown in Figure 3, a simple iterative improvement algorithm only produces the optimal 2-colouring 15% of the time.

In performing our benchmark of networks of oscillators, we implemented the Aihara *et al*. anti-phase model, applying the equations from (2) above to each pair of connected oscillators.

Figure 4 illustrates six "snap shots" from a typical simulation of the system (with K = 0.1). Snap shot (a) in the figure shows the initial state of the 32x8 configuration where the oscillators were randomly initialised to a phase. The sequence of the states as indicated in part (b), (c), (d), (e) and (f) are the states of the oscillators after 150, 300, 450, 600 and 800 oscillator cycles. The final state of the system, as shown in (f) is the optimal solution, with all oscillators in anti-phase with their neighbours.

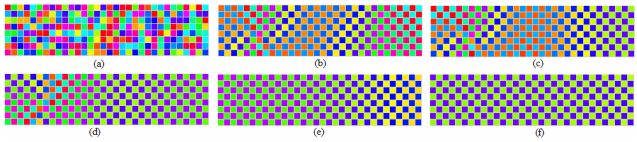


Figure 4: Stages in the convergence of the Ising model

Detailed examinations of the progress of the 32x8 oscillators demonstrate that the oscillators congregate into a number of groups and these groups slowly merge. For example, there are about 6 distinct groups in figure (b) as identifiable by the colour. The six groups begin to consolidate and increase in size in figure (c). In figure (d), two nearly synchronised groups start to dominate the right half of the network and in figure (e) the synchronised group converts the remaining oscillators on the left.

We ran the above simulation 100 times. Seventy-seven of the runs reached a global synchronised state after 1300 cycles. Investigation on the remaining twenty-three runs show that the oscillators in those runs form limit cycles. That is, the oscillators change their phases in a way that eventually brings them back to an identical set of phases; these changes then repeat. Figure 5 shows such a sequence of phases. Configuration (f) in the figure is identical to configuration (a).

Figure 5: A limit cycle in a suboptimal run.

The occurrence of limit cycles is observable during simulations, as waves or rotating spiral-like patterns as shown in the following snapshots Figure 6. The simulation in this figure consists of 1024 vertices. The patterns can be seen from early in a simulation.

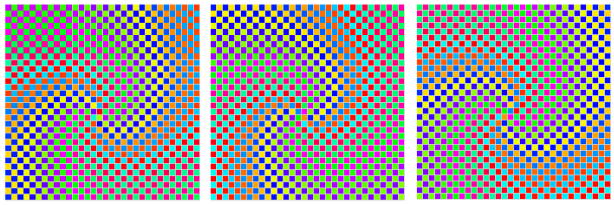


Figure 6: Rotating spirals in a simulation with 1024 vertices

3 Generalisation to a Multi-Phase Model

In order to perform an arbitrary *k*-colouring where $k > 2$, the phase coupled oscillators need to achieve a stable phase configurations with oscillators grouping into k phase-clusters. For example, for 3-fully connected oscillators, the phase difference between the oscillators should be near $2\pi/3$ (120 degrees). To admit such phase configurations, we generalised the Aihara *et al*. equations from section 1.1, as described in this section.

For a general case, where there are *n* fully connected interacting oscillators, we assume a mean field model (Kuramoto, 1984) to derive our generalisation of the Aihara *et al*. model:

$$\frac{d\theta_i}{dt} = \omega_i + \frac{K}{N}\sum_{j=0}^{N}\sin(\theta_j - \theta_i - \beta), i = 1,..., N \qquad \cdots (3)$$

The frustration parameter β and the frequency ω are the same for all the oscillators in the system.

An attractive feature of this model is that there are no parameters that need to be tuned depending upon the number of colours required by a graph.

Below, we describe an empirical study that shows this model meets our initial requirement that the angle separation should be a multiple of 1/*n* for *n* fully connected oscillators.

3.1 Testing Multi-Phase Synchrony for n>=3

Fully connected networks are a realistic scenario to explore, as nodes connected by wireless could easily implement such a completely connected network topology, with the only necessary communication among the oscillators being a broadcast of their firings.

Figure 7 shows examples of colourings for small complete graphs. The leftmost portion of the diagram shows a graph, with 3 vertices. Next to it is a colour wheel showing the phases of the three oscillators, which are spread evenly, indicating a correct colouring. Beside that is another graph, with 4 vertices. Its associated colour wheel also shows that the phases of the oscillators evenly spread.

We have tested the update equation in (3) on complete graphs, up to *n*=100 vertices (larger graphs are not practical with our simulation software). We used a coupling strength *K*/*N*=0.1. We have found that our simulations reliably converge to good solutions for a wide variety of values of β, provided $\beta \geq 2\pi/n$.

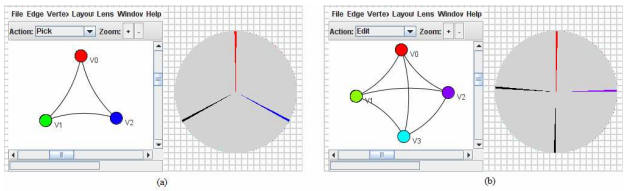


Figure 7: Angle separation in a 3 & 4 fully connected oscillators

The 3-colour problem in Figure 7 typically requires 20 oscillator cycles to reach a stable state, with a worst case of 30 oscillator cycles. The 4-colour problem in Figure 7 typically requires 50 oscillator cycles to give a reasonable phase difference between the oscillators, but requires an approximately 150 cycles to achieve a near perfect multiple $\pi/2$ phase difference.

Phase separation over a number of cycles is illustrated in Figure 8, for a complete graph of 8 nodes. The horizontal axis represents time. The vertical axis shows the phase of each of the eight oscillators when oscillator 0 fires. The phases of the oscillators are randomly initialised. Soon after the system starts, several oscillators are already in a near stable synchronisation. As the clock continues, the remaining pairs of oscillators (0 and 5) and (3 and 6) begin to separate evenly after 80 cycles. The oscillators ultimately synchronise at the 300th cycle with a near even phase difference of $\pi/4$.

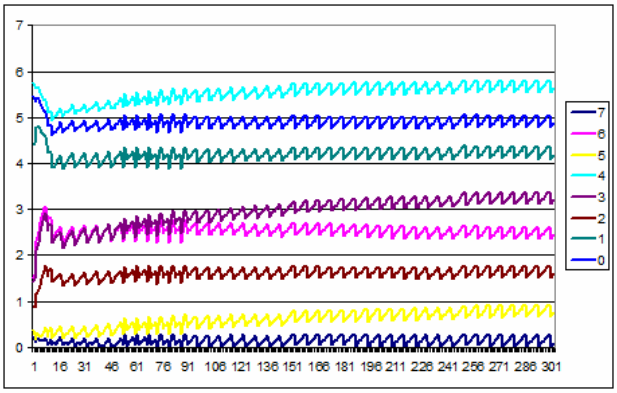


Figure 8: The phase of 8-fully connected oscillators over 300 cycles

4 Multi-phase Oscillators in Complete k-Partite Graphs

A complete *k-partite* graph has its vertices split into *k* partitions where (1) vertices in the same partitions are not connected, but (2) all nodes in each partition are connected to all nodes in the other partitions. Figure 9 illustrates k-partite graphs for k = 2, 3 and 7. Such graphs are an interesting case study to explore, as the optimal solutions are obvious (each partition requires one colour), and the results offer insight into the limitations of our generalisation of the Aihara *et al*. model.

4.1 Equal Complete k-Partite Graphs

The results from our experiment indicate that multi-phase coupled oscillators can reliably find a minimal graph colouring of complete k-partite graphs provided the number of vertices in each partition is equal. Figure 9 demonstrate colourings we have found using our update equation (3) for k-partite graphs with k=2, 3 and 7 using coupled oscillators. Colours are typically found in a small number of cycles and the system synchronises rapidly.

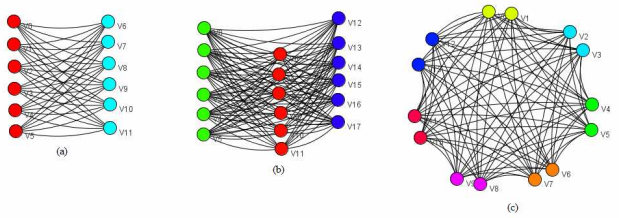


Figure 9: Complete 2, 3 and 7-partite graphs

4.2 Unequal Complete k-Partite Graphs

We performed tests where the number of vertices differs in the partitions of the complete k-partite graphs. We found that the quality of the results varies according to the size difference between partitions. Figure 10 demonstrates the colouring of k-partite graphs with unequal number of vertices in the partitions. Part (a) illustrates that good colourings can still occur if the number of oscillators in each partition is approximately equal, but part (b) demonstrates what happens as the size difference in the partitions grows.

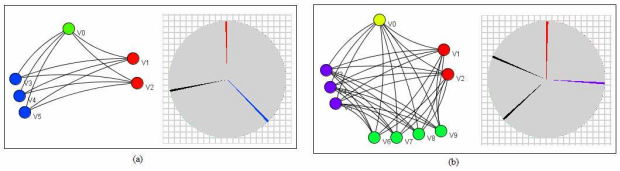


Figure 10: K-partite graphs with unequal number of vertices in each partition

4.3 A Further Illustration

A fundamental problem with colouring k-partite graphs with unequal partition sizes is more obviously illustrated on a simpler case that is not a k-partite graph. This case is illustrated in Figure 11. The graph shown can be thought of as containing two overlapping complete subgraphs of different sizes: vertices v_0-v_3 form one complete subgraph, (S1) and vertices v_3-v_5 form the other subgraph (S2). The four oscillators forming S1 tend to separate into equal phase differences corresponding to four colours (as illustrated by the leftmost colour wheel in the figure), while the three oscillators forming S2 tend to separate into equal phase differences corresponding to three colours (as illustrated by the middle colour wheel). The combined effect (as illustrated by the rightmost colour wheel) is a suboptimal solution.

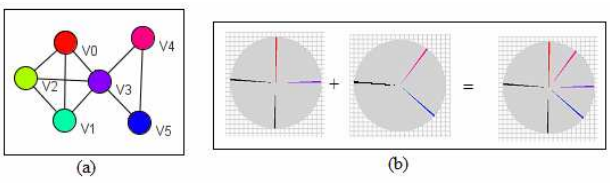


Figure 11: A simple illustration on two overlapping complete sub graphs

5 Multi-Phase Oscillators in a Plane: Experiments with Three Colours

Earlier, we examined the standard Ising Spin problem, where 2 colours are sufficient. This problem is easily generalised to forms that require 3 and 4 colours, by (for example) replacing the squares in the plane with tessellations of hexagons (3 colours) or adding extra connections to the squares so that the squares also connect diagonally (four colours).

We performed tests on four hexagonal topologies as shown in Figure 12. The top left graph in Figure 12 illustrates the simplest case for a hexagonal arrangement that can be 3-coloured. As the associated colour wheel illustrates, the oscillators forming this simple graph always synchronise with a minimum number of colours. For larger graphs, as illustrated in part (b), (c) and (d), there is an observable clustering of the oscillators phases into three groups (less obvious in part (d)), but the phases of the oscillators within those clusters remain separated.

The reason why oscillator phases remain separated is related to the reason why oscillators do not converge in k-partite graphs with unequal number of vertices in each partition. An inspection of the graphs in Figure 12 reveals that nodes in these graphs have unequal numbers of neighbours. Nodes at the periphery of the graphs can have as few as three neighbours, whereas nodes inside the graphs have as many as six neighbours. The dynamics of the update equation (3) has internal nodes and peripheral nodes having asymmetric effects on each other.

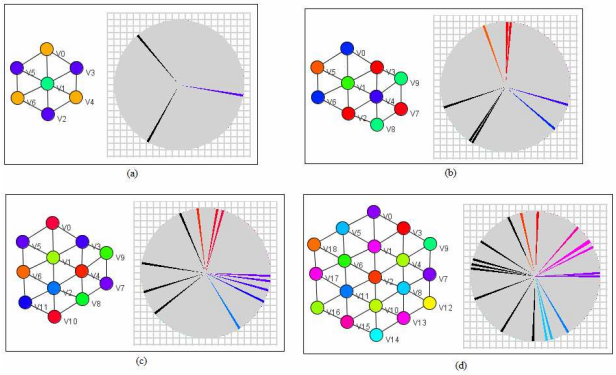


Figure 12: Colouring of hexagons

The graph colouring problem illustrated in Figure 12 scales to arbitrarily large numbers of nodes. As the size of such a network grows the ratio of peripheral elements to internal nodes decreases. Thus, the problem illustrated in Figure 12 may be less evident in large networks of computational elements.

6 Multi-phase Oscillators in Regular Graphs

The results described in the previous section, for tessellations of hexagons on a plane, show that the model does not colour the graph optimally. This suboptimal behaviour is at least partly due the unequal degree of vertices in the tessellation. For example, in Figure 12, the vertices on the fringe of the graphs typically have degree 3 or 4 while the inner vertices have degree 6.

To test whether the unequal degree of vertices completely explains such suboptimal behaviour, we performed tests on graphs where the vertices within each of the graphs have the same degree — we used graphs based on the Platonic solids and also the Ising Spin Problem on a torus.

6.1 Platonic Solids

Our experiments indicate that colourings of the first three Platonic solids, the tetrahedron, hexahedron and octahedron (all 3 and 4-regular graphs) are always optimal and are achieved in a small number of cycles. Figure 13 shows these results.

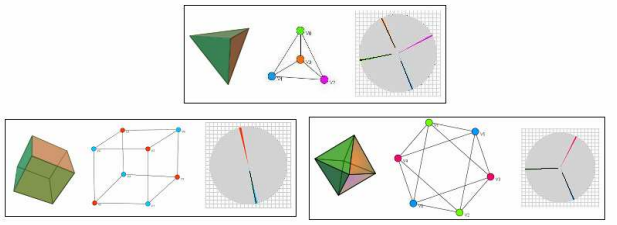


Figure 13: Colouring of simple solids

However, colourings are sub optimal for the dodecahedron (3-regular) and icosahedron (5-regular) as illustrated on Figure 14. Typically, the oscillators settle into 6 colours instead of the minimum of 3 and 4 respectively.

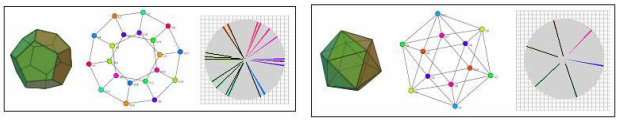


Figure 14: Colouring of dodecahedron & icosahedron

6.2 The Ising Model on a Torus

The torus formation of the Ising Spin Problem is achieved by taking a standard Euclidean Ising Spin Problem (as described in section 2), then connecting the upper and lower ends, and also the left and right ends. Consequently, all vertices have a degree of 4. This is illustrated in Figure 15. The results of 100 runs of such an 8×32 problem resulted in network convergence that was 50% faster than that of the 8×32 Euclidean Ising Spin Problem. However, only 64% of the runs attain an optimal synchronisation. Figure 15 illustrates a typical suboptimal solution, where subsets of the oscillators are optimal within their respective subsets, but the relationships between subsets is suboptimal.

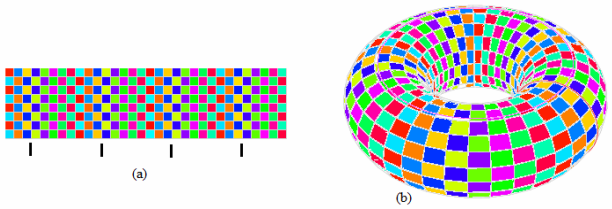


Figure 15: Sub optimal colouring on torus Ising

To illustrate this suboptimal behaviour further, we constructed a simple 1x7 Ising Spin Problem on a torus, which is a 2-regular ring graph. Ideally, the network

should converge to 3 colours. Figure 16 shows a run of such a network at 200, 400, 600 and 800 oscillator cycles. At 200 cycles, the oscillators tend to form 2 clusters at π apart. As the run continues to 800 cycles, the oscillators spread out evenly. By way of contrast, in similar experiments with rings containing an even number of oscillators, the networks always converged to an optimal 2 colouring.

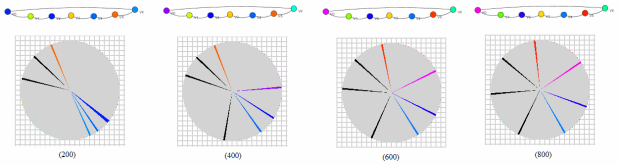


Figure 16: A simulation on a 1x7 ring

7 Characteristics of the Aihara Model

From our experiments, and as a direct consequence of the model in Equation (3), we observe the following general characteristics of networks of oscillators using our generalised Aihara model:

Observation 1: Convergence to a stable point in phase space does not imply a minimum colouring.

Observation 2: Convergence to a stable point in phase space does not imply an even phase separation among the oscillators.

Observation 3: The dynamic of the generalised Aihara equation is such that an oscillator connected to two other oscillators will move to a phase that is equidistant from the phase of the other two oscillators.

Observation 3': An oscillator O connected to two disjoint sets of oscillators, S1 and S2, where the oscillators within each set have the same phase, will move to a phase such that ratios of the phase separations from O to S1 and O to S2 will be proportional to the ratio of the sizes of S1 and S2.

Observation 4: The effectiveness of colouring graphs using networks of oscillators appears to be less dependent upon the size of the graph and more dependent on the graph topology ─ the degree of the vertices and the existence of odd or even cycles in the graph.

8 Conclusion

The purpose of carrying out this study was certainly not to find an algorithm guaranteed to minimally colour graphs – complexity theory suggests that such an algorithm does not exist. Instead, the purpose was to use graph colouring to benchmark the capacity of phase coupled oscillators to coordinate activity, in a parallel distributed fashion, within a network of simple computational elements. The results of our experiments clearly indicate that the basic oscillator phase coupling approach can effectively coordinate activity, in a parallel distributed fashion, in some types of graphs. Surprisingly, the size of graphs to be coloured is not the major determinant of effectiveness, but instead it is the topology of the graphs that most determines the effectiveness of this approach to graph colouring.

In this paper, our goal was to explore models close to the original biological source of the idea. Having identified some limitations of the pure biological approach, our future work will focus on overcoming these limitations using techniques that can be implemented in simple computational elements, without undermining the fundamental parallel, distributed nature of phase coupled oscillators. The remainder of the conclusion indicates some solutions to the problems identified in this paper.

The problem of suboptimal limit cycles, which was identified in the experiments on the standard Ising Spin problem, might be addressed by injecting a small amount of noise into the system (i.e. randomly perturbing the phase of oscillators).

A core issue to solve is the problem highlighted in the studies of k-partite graphs with unequal vertices in the partitions. One approach might be to decrease the effect of an oscillator as its phase approaches the phase of other oscillators. By a suitable formulation of the update equation, a group of closely synchronized oscillators could have the same effect on other oscillators as a single non-synchronized oscillator has on those other oscillators.

A solution to the problem highlighted in Figure 11 may only require (1) an initial global broadcast that communicates the number of colours (C) to be used in the colouring of a graph, and (2) a global agreement, via either a regular broadcasted synchronization signal, or via clocks aboard each processing element, that the final phase values of all oscillators will only differ by multiples of $2\pi/C$.

The characteristics of the Aihara model, and observation 3 in particular, indicate that a mechanism may be required whereby an oscillator can escape from having its phase trapped between the phases of two other oscillators. This might be achieved by introducing an annealing component into the way an oscillator alters its phase.

9 References

Aihara, I., Kitahata, H., Aihara, K. & Yoshikawa, K. 2006, *Periodic rhythms and anti-phase synchronization in calling behaviours of Japanese rain frogs*, University of Tokyo, http://www.i.u-tokyo.ac.jp/mi/mi-e.htm, viewed 18 September 2006.

Bennett, M., Schatz, M.F., Rockwood, H. & Wiesenfeld, K. 2002, 'Huygens' clocks', *Proceedings (A) of the Royal Society*, vol. 458, pp. 563-579.

Bettstetter, C. 2006, *Self-Organization in Communication Networks*, www.bettstetter.com/talks/bettstetter-so-2006-06-passau.pdf viewed 18 September 2006.

Buck, E. & Buck, J. 1976, 'Synchronous fireflies', *Scientific American*, no. 234, pp. 74-85.

Buck, J., Buck, E., Case, J.F. & Hanson, F.E. 1981, 'Control of flashing in fireflies. V. Pacemaker Synchronization in Pteroptyx cribellata', *Journal of comparative physiology A*, vol. 144, no. 3, pp. 287-298.

Camazine, S., Deneubourg, J.-L., Franks, N.R., Sneyd, J., Theraulaz, G. & Bonabeau, E. 2001, *Self-synchronization in biological systems*, Princeton University Press, Princeton, New Jersey.

Ermentrout, G.B. & Rinzel, J. 1984, 'Beyond a pacemaker's entrainment limit: phase walk-through', *American Journal of Physiology - Regulatory, Integrative and Comparative Physiology*, vol. 246, no. 1, pp. 102-106.

Garey, M.R. & Johnson, D.S. 1979, *Computers and Intractability: A guide to the Theory of NP-Completeness*, W. H. Freeman.

Hong, Y.W. & Scaglione, A. 2003, 'Time synchronization and reach-back communications with pulse-coupled oscillators for UWB wireless ad hoc networks', *Proceedings of the IEEE conference on Ultra Wideband Systems and Technologies*, Reston, VA, pp. 190-194.

Hopfield, J.J. & Tank, D.W. 1985, 'Neural computation of decisions in optimization problems', *Biological Cybernetics*, no. 52, pp. 141-152.

Kindermann, R. and Snell, J. L. (1980) Random Markov Fields and Their Applications, American Mathematical Society, ISBN 0-8218-3381-2.

Kuramoto, Y. 1984, *Chemical Oscillations, Waves and Turbolence*, Spinger, Berlin.

Lister, R. 1992, 'On making the right moves: Neural networks, gradient descent and simulated annealing', Unpublished PhD Thesis, Basser Department of Computer Science, The University of Sydney, Sydney.

Lucarelli, d. & Wang, I.-J. 2004, 'Decentralized Synchronization Protocols with Nearest Neighbor Communication', paper presented to the *SenSys'04*, Baltimore, Maryland, November 3-5, 2004.

Mirollo, R.E. & Strogatz, S.H. 1990, 'Synchronization of Pulse-Coupled Biological Oscillators', *SIAM Journals on Applied Mathematics*, vol. 50, no. 6, pp. 1645-1662.

Spong, M.W. 2006, 'Coordination of multi-agent systems', *The eight IASTED international conference on control and applications*, ed. C.Y. Su, Montreal, Canada, pp. 10-16.

Strogatz, S.H. 2003, *SYNC: The emerging science of spontaneous order*, Hyperion, New York.

Werner-Allen, G., Tewari, G., Patel, A., Welsh, M. & Nagpal, R. 2005, 'Firefly-Inspired Sensor Network Synchronicity with Realistic Radio Effects', paper presented to the *SenSys'05*, San Diego, California, November 2-4, 2005.

Wu, C.W. 2002, *Synchronization in coupled chaotic circuits and systems*, World Scientific Publishing Co, Singapore.

Integrating Recommendation Models for Improved Web Page Prediction Accuracy

Faten Khalil[1] **Jiuyong Li**[2] **Hua Wang**[1]

[1] Department of Mathematics & Computing
University of Southern Queensland,
Toowoomba, Australia, 4350,
Email: {khalil and wang}@usq.edu.au

[2] School of Computer & Information Science
University of South Australia,
Mason Lakes, Australia,
Email: Jiuyong.Li@unisa.edu.au

Abstract

Recent research initiatives have addressed the need for improved performance of Web page prediction accuracy that would profit many applications, e-business in particular. Different Web usage mining frameworks have been implemented for this purpose specifically Association rules, clustering, and Markov model. Each of these frameworks has its own strengths and weaknesses and it has been proved that using each of these frameworks individually does not provide a suitable solution that answers today's Web page prediction needs. This paper endeavors to provide an improved Web page prediction accuracy by using a novel approach that involves integrating clustering, association rules and Markov models according to some constraints. Experimental results prove that this integration provides better prediction accuracy than using each technique individually.

Keywords: Web page prediction, association rules, clustering, Markov model.

1 Introduction

Web page access prediction gained its importance from the ever increasing number of e-commerce Web information systems and e-businesses. Web page prediction that involves personalizing the Web users' browsing experiences assists Web masters in the improvement of the Web site structure, and helps Web users in navigating the site and accessing the information they need. Various attempts have been exploited to achieve Web page access prediction by preprocessing Web server log files and analyzing Web users' navigational patterns. The most widely used approach for this purpose is Web usage mining that entails many techniques like Markov model, association rules and clustering (Srivastava et al. 2000).

- Markov models are the most effective techniques for Web page access prediction and many researchers stress the importance in the field (Bouras & Konidaris 2004, chen et al. 2002, Deshpande & Karypis 2004, Eirinaki et al. 2005, Zhu et al. 2002). Other researchers use Markov models to improve the Web server access efficiency either by using object prefetching (Pons 2006) or by helping reduce the Web server overhead (Mathur & Apte 2007). Lower order Markov models are known for their low accuracy due to the limited availability of users' browsing history. Higher order Markov models achieve higher accuracy but are associated with higher state space complexity.
- Association rule mining is a major pattern discovery technique (Mobasher et al. 2001). The original goal of association rule mining is to solve market basket problem but the applications of association rules are far beyond that. Using association rules for Web page access prediction involves dealing with too many rules and it is not easy to find a suitable subset of rules to make accurate and reliable predictions (Kim et al. 2004, Mobasher et al. 2001, Yong et al. 2005).
- Although clustering techniques have been used for personalization purposes by discovering Web site structure and extracting useful patterns (Adami et al. 2003, Cadez et al. 2003, Papadakis & Skoutas 2005, Rigou et al. 2006, Strehl et al. 2000), usually, they are not very successful in attaining good results. Proper clustering groups users sessions with similar browsing history together, and this facilitates classification. However, prediction is performed on the cluster sets rather than the actual sessions.

Therefore, there arises a need for improvement when using any of the aforementioned techniques. This paper integrates all three frameworks together, clustering, association rules and Markov model, to achieve better Web page access prediction performance specifically when it comes to accuracy.

Web page access prediction can be useful in many applications. The improvement in accuracy can make a change in the Web advertisement area where a substantial amount of money is paid for placing the correct advertisements on Web sites. Using Web page access prediction, the right ad will be predicted according to the users' browsing patterns. Also, using the Web users' browsing patterns Web page access prediction helps Web administrators restructure the Web sites to improve site topology and user personalization as well as market segmentation. Web page access prediction is also helpful for caching the predicted page for faster access, for improved Web page ranking and for improving browsing and navigation orders.

2 Related Work

A number of researchers attempted to improve the Web page access prediction precision or coverage by combining different recommendation frameworks. For instance, many papers combined clustering with association rules (Lai & Yang 2000, Liu et al. 2001). Lai & Yang (2000) have introduced a customized marketing on the Web approach using a combination of clustering and association rules. The authors collected information about customers using forms, Web server log files and cookies. They categorized customers according to the information collected. Since k-means clustering algorithm works only with numerical data, the authors used PAM (Partitioning Around Medoids) algorithm to cluster data using categorical scales. They then performed association rules techniques on each cluster. They proved through experimentations that implementing association rules on clusters achieves better results than on non-clustered data for customizing the customers' marketing preferences. Liu et al. (2001) have introduced MARC (Mining Association Rules using Clustering) that helps reduce the I/O overhead associated with large databases by making only one pass over the database when learning association rules. The authors group similar transactions together and they mine association rules on the summaries of clusters instead of the whole data set. Although the authors prove through experimentation that MARC can learn association rules more efficiently, their algorithm does not improve on the accuracy of the association rules learned.

Other papers combined clustering with Markov model (Cadez et al. 2003, Zhu et al. 2002, Lu et al. 2005). Cadez et al. (2003) partitioned site users using a model-based clustering approach where they implemented first order Markov model using the Expectation-Maximization algorithm. After partitioning the users into clusters, they displayed the paths for users within each cluster. They also developed a visualization tool called WebCANVAS based on their model. Zhu et al. (2002) construct Markov models from log files and use co-citation and coupling similarities for measuring the conceptual relationships between Web pages. CitationCluster algorithm is then proposed to cluster conceptually related pages. A hierarchy of the Web site is constructed from the clustering results. The authors then combine Markov model based link prediction to the conceptual hierarchy into a prototype called ONE to assist users' navigation. Lu et al. (2005) were able to generate Significant Usage Patterns (SUP) from clusters of abstracted Web sessions. Clustering was applied based on a two-phase abstraction technique. First, session similarity is computed using Needleman-Wunsch alignment algorithm and sessions are clustered according to their similarities. Second, a concept-based abstraction approach is used for further abstraction and a first order Markov model is built for each cluster of sessions. SUPs are the paths that are generated from first order Markov model with each cluster of user sessions.

Combining association rules with Markov model is novel to our knowledge and only few of past researches combined all three models together (Kim et al. 2004). Kim et al. (2004) improve the performance of Markov model, sequential association rules, association rules and clustering by combining all these models together. For instance, Markov model is used first. If MM cannot cover an active session or a state, sequential association rules are used. If sequential association rules cannot cover the state, association rules are used. If association rules cannot cover the state, clustering algorithm is applied. Kim et al. (2004) work improved recall and it did not improve the Web page prediction accuracy. Our work proves to outperform previous works in terms of Web page prediction accuracy using a combination of clustering, association rules and Markov model techniques.

3 Related Methods

3.1 Clustering

This paper introduces a new model called Integrated Prediction Model, or IPM, that integrates clustering, Markov model and association rules mining frameworks in order to improve the Web page access prediction accuracy. The first problem encountered in this paper is the grouping of such sessions into k number of clusters in order to improve the Markov model prediction accuracy. Performing clustering tasks can be tedious and complex due to the increased number of clustering methods and algorithms. Clustering could be hierarchical or non-hierarchical (Jain et al. 1999), distance-based or model-based (Zhong & Ghosh 2003), and supervised or unsupervised (Eick et al. 2004). For the purpose of this paper, we use a straightforward implementation of the k-means clustering algorithm which is distance-based, based on user sessions, unsupervised and partitional non-hierarchical. K-means clustering algorithm involves the following:

1. defining a set of sessions (n-by-p data matrix) to be clustered where n represents sessions and p represents pages,
2. defining a chosen number of clusters (k) and
3. randomly assign a number of sessions to each cluster.

K-means clustering then repeatedly calculates the mean vector for all items in each cluster and reassigns the items to the cluster whose center is closest to the session until there is no change for all cluster centers. Because the first clusters are created randomly, k-means runs different times each time it starts from a different point giving different results. The different clustering solutions are compared using the sum of distances within clusters. In this paper, clusters were achieved using MatLab that considers the clustering solution with the least sum of distances. k-means clustering depends greatly on the number of clusters (k), the number of runs and the distance measure used. There exists a variety of distance measures, in particular, Euclidean, Squared Euclidean, City Block, Hamming, Cosine and Correlation (Strehl et al. 2000). In this paper we use Cosine distance measure that yields better clustering results than the other distance measures and is a direct application of the extended Jaccard coefficient (Strehl et al. 2000, Halkidi et al. 2003, Casale 2005).

3.2 Markov Model

After dividing user sessions into a number of clusters using cosine distance measure, Markov model analysis are carried out on each of the clusters. Markov models are used in the identification of the next page to be accessed by the Web site user based on the sequence of previously accessed pages (Deshpande & Karypis 2004). Let $P = \{p1, p2, \ldots, pm\}$ be a set of pages in a Web site. Let W be a user session including a sequence of pages visited by the user in a visit. Assuming that the user has visited l pages, then prob$(p_i|W)$ is the probability that the user visits pages p_i next. Page p_{l+1} the user will visit next is estimated by:

$$P_{l+1} = \text{argmax}_{p \in \mathbb{P}}\{P(P_{l+1} = p|W)\}$$
$$= \text{argmax}_{p \in \mathbb{P}}\{P(P_{l+1} = p|p_l, p_{l-1}, \ldots, p_1)\} \quad (1)$$

This probability, $prob(p_i|W)$, is estimated by using all sequences of all users in history (or training data), denoted by W. Naturally, the longer l and the larger W, the more accurate $prob(p_i|W)$. However, it is infeasible to have very long l and large W and it leads to unnecessary complexity. Therefore, a more feasible probability is estimated by assuming that the sequence of the Web pages visited by users follows a Markov process that imposes a limit on the number of previously accessed pages k. In other words, the probability of visiting a page p_i does not depend on all the pages in the Web session, but only on a small set of k preceding pages, where $k << l$.

The equation becomes:

$$P_{l+1} = \text{argmax}_{p \in \mathbb{P}}\{P(P_{l+1} = p|p_l, p_{l-1}, \ldots, p_{l-(k-1)})\} \quad (2)$$

where k denotes the number of the preceding pages and it identifies the order of the Markov model. The resulting model of this equation is called the all k^{th} order Markov model. Of course, the Markov model starts calculating the highest probability of the last page visited because during a Web session, the user can only link the page he is currently visiting to the next one. The probability of $P\left(p_i|S_j^k\right)$ is estimated as follows from a history (training) data set.

$$P\left(p_i|S_j^k\right) = \frac{\text{Frequency}\left(\left\langle S_j^k, p_i\right\rangle\right)}{\text{Frequency}\left(S_j^k\right)} \quad . \quad (3)$$

This formula calculates the conditional probability as the ratio of the frequency of the sequence occurring in the training set to the frequency of the page occurring directly after the sequence.

The fundamental assumption of predictions based on Markov models is that the next state is dependent on the previous k states. The longer the k is, the more accurate the predictions are. However, longer k causes the following two problems: The coverage of model is limited and leaves many states uncovered; and the complexity of the model becomes unmanageable (Deshpande & Karypis 2004). Therefore, the following are three modified Markov models for predicting Web page access.

1. All k^{th} Markov model: This model is to tackle the problem of low coverage of a high order Markov model. For each test instance, the highest order Markov model that covers the instance is used to predict the instance. For example, if we build an all 4-Markov model including 1-, 2-, 3-, and 4-, for a test instance, we try to use 4-Markov model to make prediction. If the 4-Markov model does not contain the corresponding states, we then use the 3-Markov model, and so forth (Pitkow & Pirolli 1999).

2. Frequency pruned Markov model: Though all k^{th} order Markov models result in low coverage, they exacerbate the problem of complexity since the states of all Markov models are added up. Note that many states have low statistically predictive reliability since their occurrence frequencies are very low. The removal of these low frequency states affects the accuracy of a Markov model. However, the number of states of the pruned Markov model will be significantly reduced.

3. Accuracy pruned Markov model: Frequency pruned Markov model does not capture factors that affect the accuracy of states. A high frequent state may not present accurate prediction. When we use a means to estimate the predictive accuracy of states, states with low predictive accuracy can be eliminated. One way to estimate the predictive accuracy using conditional probability is called confidence pruning. Another way to estimate the predictive accuracy is to count (estimated) errors involved, called error pruning.

In this paper, we employ the frequency pruned Markov model. When choosing the Markov model order, our aim is to determine a Markov model order that leads to high accuracy with low state space complexity. Figure 1 reveals the increase of precision as the frequency pruned Markov model increases using the four data sets introduced in section 5 below. On the other hand, table 1 and table 2 show the increase of the state space complexity as the order of all k^{th} and frequency pruned Markov model increases for all four data sets. The frequency pruned Markov model orders, and as it has been proposed by (Deshpande & Karypis 2004), does not increase the prediction accuracy significantly. It rather plays a major role in decreasing the state space complexity. Based on this information, we use the 2-FP order Markov model because it has better accuracy than that of the 1-FP order Markov model without the drawback of the state space complexity associated with higher order Markov models.

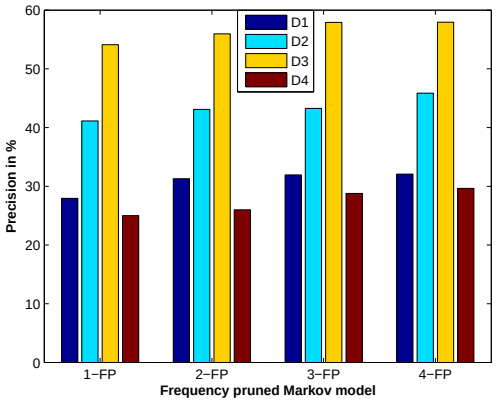


Figure 1: Precision of all 1-, 2-, 3- and 4- frequency pruned Markov model orders.

Table 1: Number of states of all 1- to 4- Markov model orders.

	1-MM	2-MM	3-MM	4-MM
D1	1945	39162	72524	101365
D2	1036	25060	89815	128516
D3	674	21392	50971	83867
D4	2054	34469	90123	131106

Table 2: Number of states of frequency pruned Markov model orders.

	1-FP	2-FP	3-FP	4-FP
D1	745	9162	14977	17034
D2	502	6032	18121	22954
D3	623	5290	11218	13697
D4	807	7961	19032	23541

3.3 Association Rules

The final step in the training process is to generate global association rules from the original data. Association rules are mainly defined by two metrics: support and confidence. Let A be a subsequence of W, and p_i be a page. We say that W supports A if A is a subsequence of W, and W supports $\langle A, p_i \rangle$ if $\langle A, p_i \rangle$ is a subsequence of W. The support for sequence A is the fraction of sessions supporting A in the data set D as follows:

$$\sigma = supp(A) = \frac{\{W \in D : A \subseteq W\}}{D} \quad (4)$$

The confidence of the implication is:

$$\alpha = conf(A) = \frac{supp(\langle A, P \rangle)}{supp(A)} \quad (5)$$

An implication is called an association rule if its support and confidence are not less than some user specified minimum thresholds. The selection of parameter values for σ and α usually has to be based on experience or even resorts to try and error. The most common association mining algorithm is Apriori algorithm (Agrawal & Srikant 1994). The main problem of mining association rules is composed of two steps:

1. Discovery of large itemsets.
2. Using the large itemsets to generate the association rules.

The second step is simple and the overall performance of mining association rules is determined by the first step. Apriori (Agrawal & Srikant 1994) addresses the issue of discovering large itemsets. In each iteration, Apriori constructs a candidate set of large itemsets, counts the number of occurrences of each candidate and determines the large itemsets based on a predetermined minimum support and confidence thresholds. In the first iteration, Apriori scans all the transactions to count the number of occurrences for each item and based on the minimum support threshold (σ), the first large itemset is determined. Therefore, the cost of the first iteration is $O(D)$. Next, the second large itemset is determined by concatenating items in the first large itemset and applying the minimum support test to the results. More iterations will take place until there are no more candidate itemsets. In simple terms, the cost of the algorithm is $O(I * D)$ where I denotes the number of iterations used. Association rules are generated based on all large itemsets. The generated rules are so large and complex that they can lead to conflicting results.

The Apriori algorithm is usually implemented on large data sets where the items within the one transaction are not in any particular order. This contradicts Web data sets where the pages are accessed in a particular order. Therefore, there was a need to implement sequential association rules using the Apriori algorithm. There are four types of sequential association rules presented by Yang et al. (2004):

1. Subsequence rules: they represent the sequential association rules where the items are listed in order.
2. Latest subsequence rules: They take into consideration the order of the items and most recent items in the set.
3. Substring rules: They take into consideration the order and the adjacency of the items.
4. Latest substring rules: They take into consideration the order of the items, the most recent items in the set as well as the adjacency of the items.

In this paper, we will use sequential association rule mining on user transaction data to discover Web page usage patterns. Prediction of the next page to be accessed by the user is performed by matching the discovered patterns against the user sessions. This is usually done online.

4 Proposed Model

4.1 Motivation for the Combined Approach

Our work is based on combining clustering algorithm, association rules mining and Markov model during the prediction process. The IPM integration during the prediction process is novel and proves to outperform each individual prediction model mentioned in section 1 as well as the different combination models addressed in section 2. The IPM integration model improves the prediction accuracy as opposed to other combinations that prove to improve the prediction coverage and complexity. The improvement in accuracy is based on different constraints like dividing the data set into a number of clusters based on services requested by users. This page categorization method proves to yield better clustering results (Wang et al. 2004). Therefore, better clusters means better Markov model prediction accuracy because the Markov model prediction will be based on more meaningfully grouped data. It also improves the state space complexity because Markov model prediction will be carried out on one particular cluster as opposed to the whole data set. The other constraint is using association rules mining in the case of a state absence in the training data or where the state prediction probability is not marginal. This helps improve the prediction accuracy because association rules look at more history and examine more states than Markov models. Also, IPM will not be subjected to the complexity associated with the number of rules generated because the rules will be examined in special cases only. Another constraint is the distance measure used in the identification of the appropriate cluster that each new page should belong to. The cosine distance measure has proved to outperform other distance measures like Euclidean, hamming, correlation and city block (Strehl et al. 2000, Halkidi et al. 2003). The prediction accuracy based on the integration of the three frameworks together according to these constraints proves to outperform the prediction accuracy based on each of the frameworks individually.

4.2 Algorithm

The process is as follows:

```
Training:

(1)Combine functionally related pages according
   to services requested

(2)Cluster user sessions into l-clusters

(3)Build a k-Markov model for each cluster

(4) For Markov model states where the
    majority is not clear

(5)  Discover association rules for each
     state

(6)EndFor
```

Combining similar pages or allocating related pages to categories is an important step in the training process of the IPM model. Consider a data set D containing N number of sessions. Let W be a user session including a sequence of pages visited by the user in a visit. $D = \{W_1, ..., W_N\}$. Let $P = \{p1, p2, \ldots, pm\}$ be a set of pages in a Web site. Since Markov model techniques will be implemented on the data, the pages have to remain in the order by which they were visited. $W_i = (p_1^i, ..., p_L^i)$ is a session of length L composed of multivariate feature vectors p. The set of pages P is divided into a number of categories C_i where $C_i = \{p1, p2, \ldots, pn\}$. This results in less number of pages since $C_i \subset P$ and $n < m$. For each session, a binary representation is used assuming each page is either visited or not visited. If the page is visited, a weight factor w is added to the pages representing the number of times the page was visited in the new session S_i. $S_i = \{(c_1^i, w_1^i), ..., (c_L^i, w_j^i)\}$. Ds is the data set containing N number of sessions S_N.

The categories are formed as follows:

Input: D containing N number of sessions W_N.

(1) For each page p_i in session W_i

(2) If $p_i \subset C_i$

(3) w_i.count++

(4) Else,

(5) $w_i = 0$

(6) EndIf

(7) EndFor

Output: Ds containing N number of Sessions S_N.

Combining the similar Web pages into categories Ci, increases the value of w and makes all sessions of equal length. According to Casale (2005), sessions of equal length give better similarity measures results. As an example, consider the following three sessions:

W1	1, 2, 3, 1, 3
W2	1, 2, 1,
W3	3, 1, 3

If pages 1 and 2 belong to category1 and page 3 belongs to category2, we have the following sessions:

Category	1	2
S1	3	2
S2	3	0
S3	1	2

Clustering the resulting sessions S_N was implemented using k-means clustering algorithm according to the Cosine distance between the sessions. Consider two sessions Sa and Sb. The Cosine distance between Sa and Sb is given by:

$$\text{distCosine}(Sa, Sb) = \frac{\sum(Sa_i Sb_i)}{\sqrt{\sum(Sa_i)^2}\sqrt{\sum(Sb_i)^2}} \quad (6)$$

Table 3 has 4 sessions with 4 pages each. If we are to form two clusters with two sessions each, we have to measure the distances between the sessions.

Table 3: Sessions

S1	3, 0, 5, 1
S2	2, 0, 5, 0
S3	0, 5, 0, 4
S4	0, 3, 0, 3

Table 4: Sessions distances

distCosine($S1, S2$)	0.019
distCosine($S1, S3$)	0.89
distCosine($S2, S3$)	1.0
distCosine($S1, S4$)	0.88
distCosine($S3, S4$)	0.06

Table 4 reveals the distances calculated using equation 1:

Clusters are formed according to the least distances between sessions, or the closest distances between sessions. Therefore, $\{S1, S2\}$ will form a cluster and $\{S3, S4\}$ will form another cluster.

```
Prediction:

(1)For each coming session

(2)   Find its closest cluster

(3)   Use corresponding Markov model to make
      prediction

(4)   If the predictions are made by states
      that do not belong to a majority class

(5)      Use association rules to make a
         revised prediction

(6)   EndIf

(7)EndFor
```

During the prediction process, each new page is examined and the appropriate cluster the new test point belongs to is identified. Let p_t be a new test point where $p_t \subset P$. Web sessions W are divided into K groups or clusters. The new point p_t has probability $prob(x_i = k)$ of belonging to cluster k where $\sum_k prob(x_i = k) = 1$ and x_i indicates the cluster membership of the new point p_t. The actual cluster k that the point p_t belongs to depends on the minimum distance of p_t to the mean values of K cluster centroids using the Cosine distance measure as follows:

$$\text{distCosine}(p_t, \mu) = \frac{\sum_{k=1}^{K}(p_t \mu)}{\sqrt{\sum_{k=1}^{K}(p_t)^2}\sqrt{\sum_{k=1}^{K}(\mu)^2}} \quad (7)$$

To continue with the prediction process, Markov model prediction is performed on the new identified cluster. If the Markov model prediction results in no state or a state that does not belong to the majority class, association rules mining is used instead. The majority class includes states with high probabilities where probability differences between two pages are significant. On the other hand, the minority class includes all other cases. In particular, the minority class includes:

1. States with high probabilities where probability differences between two pages are below (ϕ_c) or equal to zero.

2. States where test data does not match any of the Markov model outcomes.

A Markov model state is retained only if the probability difference between the most probable state and the second probable state is above (ϕ_c) (Deshpande & Karypis 2004). Another important issue here is defining the majority class and identifying whether the new state belongs to the majority or the minority class. This in mind, we employ the confidence pruned Markov model introduced by Deshpande *et. al*, (Deshpande & Karypis 2004). The confidence threshold is calculated as follows:

$$\phi_c = \hat{p} - z_{\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{n}} \quad (8)$$

Where $z_{\alpha/2}$ is the upper $\alpha/2$ percentage point of the standard normal distribution, and n is the frequency of the Markov state. Equation 5 stresses out the fact that states with high frequency would lead to smaller confidence threshold. That means that even if the difference between the two most probable pages is small, the state with higher probability will be chosen in the case of high frequency of the state occurrence. The smaller confidence threshold results in larger majority class. The effect of the confidence threshold value and, therefore, the majority class size on the prediction accuracy depends on the actual data set. To determine the optimal value of $z_{\alpha/2}$ and, as a result, the value of the confidence factor ϕ_c we conducted an experiment using the EPA data set (later referred to as D1 and described in section 6). As Table 5 depicts, the increase of the minority class or, in other words, the increase in the confidence factor is affected by the decrease of $z_{\alpha/2}$. During the prediction process, if the Markov model probability belongs to the minority class, association rules probability for the item is taken into consideration instead. Table 3 displays the results of the IPM accuracy using different values for $z_{\alpha/2}$. It is clear that the accuracy increases at first with lower confidence threshold and therefore, larger minority class. However, after a certain point, accuracy starts to decrease when the majority class is reduced to the extent where it looses the advantage of the accuracy obtained by combining Markov model and clustering. The optimal value for $z_{\alpha/2}$ is 1.15. Note that the number of states has dramatically decreased.

Table 5: Accuracy according to $z_{\alpha/2}$ value

$z_{\alpha/2}$	Accuracy	# states
0	31.29	9162
0.75	33.57	2061
0.84	35.45	1932
0.93	37.80	1744
1.03	40.60	1729
1.15	44.91	1706
1.28	43.81	1689
1.44	40.93	1614
1.64	38.85	1557
1.96	37.91	1479
2.57	36.81	1304

With $z_{\alpha/2}$=1.15, the most probable pages range approximately between 80% and 40% with ϕ_c ranging between 47% and zero respectively given n=2. This results in approximately 0.78 as the ratio of the majority class to the whole data set. This leaves space for 22% improvement using association rules mining not including instances that have zero matching states in the training data set.

4.3 Example

Consider table 6 that depicts data transactions performed by a user browsing a Web site.

Table 6: User sessions

T1	A,F,I,J,E,C,D,H,N,I,J,G,D,H,N,C,I,J,G
T2	F,D,H,N,I,J,E,A,C,D,H,N,I,J,G
T5	E,C,A,C,F,I,A,C,G,A,D,H,M,G,J
T3	F,D,H,I,J,E,H,F,I,J,E,D,H,M
T4	G,E,A,C,F,D,H,M,I,C,A,C,G

Performing clustering analysis on the data set using k-means clustering algorithm and Cosine distance measure where the number of clusters k=2 results in the following two clusters:

Cluster 1:

T1	A,F,I,J,E,C,D,H,N,I,J,G,D,H,N,C,I,J,G
T2	F,D,H,N,I,J,E,A,C,D,H,N,I,J,G
T3	F,D,H,I,J,E,H,F,I,J,E,D,H,M

Cluster 2:

T5	E,C,A,C,F,I,A,C,G,A,D,H,M,G,J
T4	G,E,A,C,F,D,H,M,I,C,A,C,G

Consider the following test data state I $\rightarrow$ J $\rightarrow$?. Applying the 2^{nd} order Markov Model to the above training user sessions we notice that the state $\langle I, J\rangle$ belongs to cluster 1 and it appeared 7 times as follows:

$$P_{l+1} = \text{argmax}\{P(E|J,I)\} = \text{argmax}\{E \rightarrow 0.57\}$$

$$P_{l+1} = \text{argmax}\{P(G|J,I)\} = \text{argmax}\{G \rightarrow 0.43\}$$

This information alone does not provide us with correct prediction of the next page to be accessed by the user as we have high probabilities for both pages, G and E. Although the result does not conclude with a tie, neither G nor E belong to the majority class. The difference between the two pages (0.14), is not higher than the confidence threshold (in this case 0.2745). In order to find out which page would lead to the most accurate prediction, we have to look at previous pages in history. This is where we use subsequence association rules as it appears in Table 7 below.

Table 7: User sessions history

A, F,	$\langle I, J\rangle$	E
C, D, H, N,	$\langle I, J\rangle$	G
D, H, N, C,	$\langle I, J\rangle$	G
F, D, H, N,	$\langle I, J\rangle$	E
A, C, D, H, N,	$\langle I, J\rangle$	G
F, D, H,	$\langle I, J\rangle$	E
H, F,	$\langle I, J\rangle$	E

Table 8 and Table 9 summarise the results of applying subsequence association rules to the training

Table 8: Confidence of accessing page E using subsequence association rules

A → E	AE/A	1/2	50%
F → E	FE/F	4/4	100%
D → E	DE/D	2/6	33%
H → E	HE/H	2/7	29%
N → E	NE/N	1/4	25%

Table 9: Confidence of accessing page G using subsequence association rules

C → G	CG/C	3/3	100%
D → G	DG/D	3/6	50%
H → G	HG/H	3/7	43%
N → G	NG/N	3/4	75%
A → G	AG/A	1/2	50%

data. Table 8 shows that F → E has the highest confidence of 100%. While Table 9 shows that C→ G has the highest confidence of 100%.

Using Markov models, we can determine that the next page to be accessed by the user after accessing the pages I and J could be either E or G. Whereas subsequence association rules take this result a step further by determining that if the user accesses page F before pages I and J, then there is a 100% confidence that the user will access page E next. Whereas, if the user visits page C before visiting pages I and J, then there is a 100% confidence that the user will access page G next.

5 Experimental Evaluation

In this section, we present experimental results to evaluate the performance of our algorithm. All experiments were conducted on a P4 1.8 GH PC with 1GB of RAM running Windows XP Professional. The algorithms were implemented using MATLAB.

For our experiments, the first step was to gather log files from active web servers. Usually, Web log files are the main source of data for any e-commerce or Web related session analysis (Spiliopoulou et al. 1999). The logs are an ASCII file with one line per request, with the following information: The host making the request, date and time of request, requested page, HTTP reply code and bytes in the reply. The first log file used is a day's worth of all HTTP requests to the EPA WWW server located at Research Triangle Park, NC. The logs were collected for Wednesday, August 30 1995. There were 47,748 total requests, 46,014 GET requests, 1,622 POST requests, 107 HEAD requests and 6 invalid requests. The second log file is SDSC-HTTP that contains a day's worth of all HTTP requests to the SDCS WWW server located at the San Diego Supercomputer Center in San Diego, California. The logs were collected from 00:00:00 PDT through 23:59:41 PDT on Tuesday, August 22 1995. There were 28,338 requests and no known losses. The third log file is CTI that contains a random sample of users visiting the CTI Web site for two weeks in April 2002. There were 115,460 total requests. The fourth log file is Saskatchewan-HTTP which contains one week worth of all HTTP requests to the University of Saskatchewan's WWW server. The log was collected from June 1, 1995 through June 7, 1995, a total of seven days. In this one week period there were 44,298 requests.

Before using the log files data, it was necessary to perform data preprocessing (Zhao et al. 2005, Sarukkai 2000). We removed erroneous and invalid pages. Those include HTTP error codes 400s, 500s, and HTTP 1.0 errors, as well as, 302 and 304 HTTP errors that involve requests with no server replies. We also eliminated multi-media files such as gif, jpg and script files such as js and cgi.

Next step was to identify user sessions. A session is a sequence of URLs requested by the same user within a reasonable time. The end of a session is determined by a 30 minute threshold between two consecutive web page requests. If the number of requests is more than the predefined threshold value, we conclude that the user is not a regular user; it is either a robot activity, a web spider or a programmed web crawler. The sessions of the data sets are of different lengths. They were represented by vectors with the number of occurrence of pages as weights.

Table 10 represents the different data sets after preprocessing.

Table 10: Sessions

	D1	D2	D3	D4
# Requests	47,748	28,338	115,460	44,298
# Sessions	2,520	4,356	13,745	5,673
# Pages	3,730	1,072	683	2,385
# Unique IPs	2,249	3,422	5,446	4,985

Further preprocessing of the Web log sessions took place by removing short sessions and only sessions with at least 5 pages were considered. This resulted in further reducing the number of sessions. Finally, sessions were categorized according to feature selection techniques introduced by Wang et al. (Wang et al. 2004). The pages were grouped according to services requested which yield best results if carried out according to functionality (Wang et al. 2004). This could be done either by removing the suffix of visited pages or the prefix. In our case, we could not merge according to suffix because, for example, pages with suffix index.html could mean any default page like OWOW/sec4/index.html or OWOW/sec9/index.html or ozone/index.html. Therefore, merging was according to a prefix. Since not all Web sites have a specific structure where we can go up the hierarchy to a suitable level, we had to come up with a suitable automatic method that can merge similar pages automatically. A program runs and examines each record. It only keeps the delimited and unique word. A manual examination of the results also takes place to further reduce the number of categories by combining similar pages.

5.1 Clustering, Markov Model and Association Rules

All clustering experiments were developed using MATLAB statistics toolbox. Since k-means computes different centroids each run and this yields different clustering results each time, the best clustering solution with the least sum of distances is considered using MATLAB k-means clustering solutions. Therefore, using Cosine distance measure with the number of clusters (k)=7 leads to good clustering results while keeping the number of clusters to a minimum.

Merging Web pages by web services according to functionality reduces the number of unique pages and, accordingly, the number of sessions. The categorized sessions were divided into 7 clusters using the k-means algorithm and according to the Cosine distance measure.

Markov model implementation was carried out for the original data in each cluster. The clusters were divided into a training set and a test set each and 2-Markov model accuracy was calculated accordingly. Then, using the test set, each transaction was considered as a new point and distance measures were calculated in order to define the cluster that the point belongs to. Next, 2-Markov model prediction accuracy was computed considering the transaction as a test set and only the cluster that the transaction belongs to as a training set.

Since association rules techniques require the determination of a minimum support factor and a confidence factor, we used the experimental data to help determine such factors. We can only consider rules with certain support factor and above a certain confidence threshold.

Using the D1, or EPA, data set, Figure 2 below shows that the number of generated association rules dramatically decreases with the increase of the minimum support threshold with a fixed 90% confidence factor. Reducing the confidence factor results in an increase in the number of rules generated. This is apparent in Figure 3 where the number of generated rules decreases with the increase of the confidence factor while the support threshold is a fixed 4% value. It is also apparent from Figure 2 and Figure 3 below that the influence of the minimum support factor is much greater on the number of rules than the influence of the confidence factor. The association rules precision is calculated as a fraction of correct recommendations to total test cases used.

$$Precision(Te) = \frac{Te \cap Tr}{Te} \qquad (9)$$

Te represents the test cases whereas Tr represents training test cases or (D-Te).

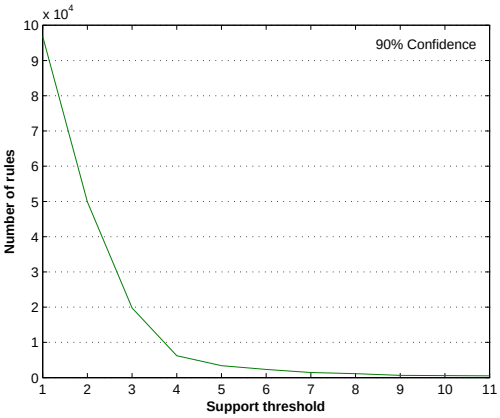


Figure 2: Number of rules generated according to different support threshold values and a fixed confidence factor: 90%.

Larger minimum support means less number of rules but it could also mean that genuine rules might be omitted. Figure 4 depicts the time complexity of generating association rules using different values of σ for D1 data set.

5.2 Experiments Results

Figure 5 depicts better Web page access prediction accuracy by integrating Markov model, Association rules and clustering (IPM). Prediction accuracy was computed as follows:

1. The data set is clustered according to k-means clustering algorithm and Cosine distance measure.

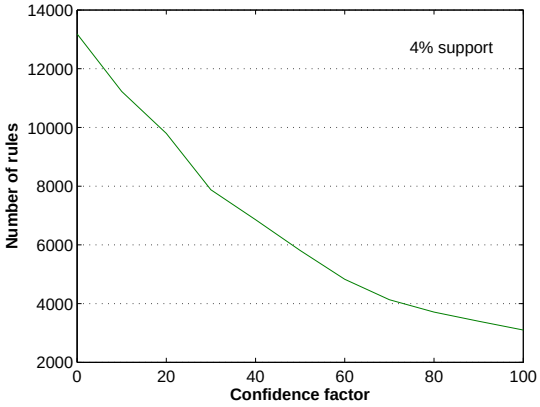


Figure 3: No. of rules generated according to a fixed support threshold: 4%.

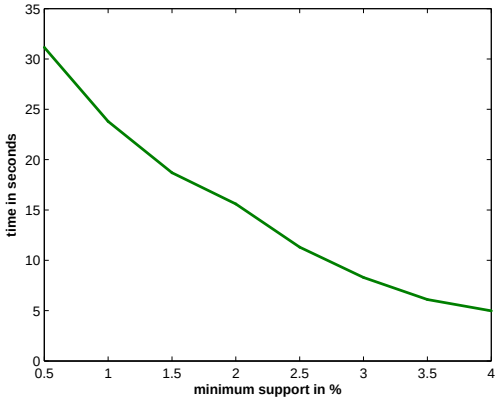


Figure 4: Time complexity in seconds for different support value.

2. For each new instance, the prediction accuracy is calculated based on the 2-MM performed on the closest cluster.
3. If the prediction results in a state that does not belong to the majority class, global association rules are used for prediction.
4. The frequency of the item is also determined in that particular cluster.
5. ϕ_c is calculated for the new instance using $z_{\alpha/2}$ value to determine if it belongs to the majority class.
6. if the state does not belong to the majority class, global association rules are used to determine the prediction accuracy, otherwise, the original accuracy is used.

Figure 5 shows that IPM results in better prediction accuracy than any of the other techniques individually. It also reveals that the increase in accuracy depends on the actual data set used. For instance, D4 prediction accuracy was increased while using a combination of MM and AR than by combining MM and clustering. On the other hand, D2 experienced more increase in accuracy using MM and clustering than using MM and AR. The accuracy increase of D1 and D3 was somewhat constant. Prediction accuracy results were achieved using the maximum likelihood based on conditional probabilities as stated in equation 4 above. All predictions in the test data that did not exist in the training data sets were assumed

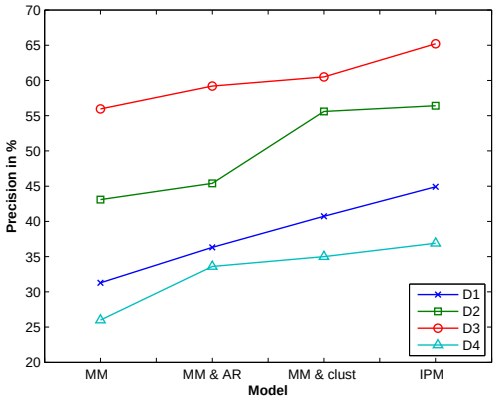


Figure 5: Precision of Markov model (MM) and MM with Association rules mining and MM with Clustering and all three models together (IPM) .

incorrect and were given a zero value. The Markov model accuracy was calculated using a 10-fold cross validation. The data was partitioned into T for testing and $(D - T)$ for training where D represents the data set. This procedure was repeated 10 times, each time T is moved by T number of transactions. The mean cross validation was evaluated as the average over the 10 runs. Table 11 reveals the standard deviation of all mean values of prediction accuracy for all four data sets.

Table 11: Accuracy values standard deviation

	D1	D2	D3	D4
MM	4.69	3.90	2.71	1.36
MM + AR	3.07	1.98	5.32	2.17
MM + Clust	2.55	2.94	1.45	3.83
IPM	1.32	3.07	6.19	2.69

The standard deviation results are considerably low compared to the mean values. This means that MM, MM + AR, MM + Clust and IPM accuracy results are quite different from each other lying on an improved baseline. The low standard deviation figures give more weight and significance to the improved prediction accuracy displayed in figure 5 above.

5.3 IPM Efficiency Analysis

All clustering runs were performed on a desktop PC with a Pentium IV Intel processor running at 2 GHz with 2 GB of RAM and 100 GB of hard disk memory. The runtime of the k-means algorithm, regardless of the distance measure used, is equivalent to O(nkl) (Jain et al. 1999), where n is the number of items, k is the number of clusters and l is the number of iterations taken by the algorithm to converge. For our experiments, where n and k are fixed, the algorithm has a linear time complexity in terms of the size of the data set. The k-means algorithm has a $O(k + n)$ space complexity. This is because it requires space to store the data matrix. It is feasible to store the data matrix in a secondary memory and then the space complexity will become O(k). k-means algorithm is more time and space efficient than hierarchical clustering algorithms with $O(n^2 logn)$ time complexity and $O(n^2)$ space complexity. As for all 2^{nd} order Markov model, the running time of the whole data set was similar to that of the clusters added together because the running time is in terms of the size of the data. i.e. T(n)=T(k1)+T(k2)+T(k3)+...T(ki) where time is denoted by T, the number of items in the data set is denoted by n, and the clusters are denoted by ki. The running time of association rule mining is O(I.D) as explained above. The association rules produced were for the whole data set. Accessing the appropriate rule is, however, performed online at time of prediction.

Constructing the IPM model is more complex than the individual models as it involves constructing k-means clustering, Markov model and association rules for the whole data sets. However, the IPM model prediction complexity is reduced due to the fact that the prediction process involves retrieving Markov models of one cluster as opposed to the whole data set. This reduces the running time by around 85%. Also, association rules are only retrieved in the case where the state does not belong to the majority class. This gives the conclusion that the complexity of IPM depends on the size of the majority class. Larger majority class yields less complex prediction as it involves less association rules accesses. However, larger majority class does not leave a larger room for accuracy improvement.

6 Conclusion

This paper improves the Web page access prediction accuracy by integrating all three prediction models: Markov model, Clustering and association rules according to certain constraints. Our model, IPM, integrates the three models using 2-Markov model computed on clusters achieved using k-means clustering algorithm and Cosine distance measures for states that belong to the majority class and performing association rules mining on the rest. The IPM model could be extended to a completely "hands-off" or automated system. Currently, some human intervention is required especially during the features selection process.

7 Acknowledgement

This work has been partially supported by ARC Discovery Grant DP0774450 to Li and Wang.

References

Adami, G., Avesani, P. & Sona, D. (2003), 'Clustering documents in a web directory', *WIDM'03, USA* pp. 66–73.

Agrawal, R. & Srikant, R. (1994), 'Fast algorithms for mining association rules', *VLDB'94, Chile* pp. 487–499.

Bouras, C. & Konidaris, A. (2004), 'Predictive prefetching on the web and its potential impact in the wide area', *WWW: Internet and Web Information Systems* (7), 143–179.

Cadez, I., Heckerman, D., Meek, C., Smyth, P. & White, S. (2003), 'Model-based clustering and visualization of navigation patterns on a web site', *Data Mining and Knowledge Discovery* **7**.

Casale, G. (2005), 'Combining queueing networks and web usage mining techniques for web performance analysis', *ACM Symposium on Applied Computing* pp. 1699–1703.

chen, M., LaPaugh, A. S. & Singh, J. P. (2002), 'Predicting category accesses for a user in a structured information space', *SIGIR'02, Finland* pp. 65–72.

Deshpande, M. & Karypis, G. (2004), 'Selective markov models for predicting web page accesses', *Transactions on Internet Technology* **4**(2), 163–184.

Eick, C. F., Zeidat, N. & Zhao, Z. (2004), 'Supervised clustering - algorithms and benefits', *IEEE ICTAI'04* pp. 774–776.

Eirinaki, M., Vazirgiannis, M. & Kapogiannis, D. (2005), 'Web path recommendations based on page ranking and markov models', *WIDM'05* pp. 2–9.

Halkidi, M., Nguyen, B., Varlamis, I. & Vazirgiannis, M. (2003), 'Thesus: Organizing web document collections based on link semantics', *The VLDB Journal* **2003**(12), 320–332.

Jain, A. K., Murty, M. N. & Flynn, P. J. (1999), 'Data clustering: A review', *ACM Computing Surveys* **31**(3), 264–323.

Kim, D., Adam, N., Alturi, V., Bieber, M. & Yesha, Y. (2004), 'A clickstream-based collaborative filtering personalization model: Towards a better performance', *WIDM '04* pp. 88–95.

Lai, H. & Yang, T. C. (2000), 'A group-based inference approach to customized marketing on the web - integrating clustering and association rules techniques', *Hawaii International Conference on System Sciences* pp. 37–46.

Liu, F., Lu, Z. & Lu, S. (2001), 'Mining association rules using clustering', *Intelligent Data Analysis* (5), 309–326.

Lu, L., Dunham, M. & Meng, Y. (2005), 'Discovery of significant usage patterns from clusters of clickstream data', *WebKDD '05* .

Mathur, V. & Apte, V. (2007), 'An overhead and resource contention aware analytical model for overloaded web servers', *WOSP'07, Argentina* .

Mobasher, B., Dai, H., Luo, T. & Nakagawa, M. (2001), 'Effective personalization based on association rule discovery from web usage data', *WIDM'01, USA* pp. 9–15.

Papadakis, N. K. & Skoutas, D. (2005), 'STAVIES: A system for information extraction from unknown web data sources through automatic web warpper generation using clustering techniques', *IEEE Transactions on Knowledge and Data Engineering* **17**(12), 1638–1652.

Pitkow, J. & Pirolli, P. (1999), 'Mining longest repeating subsequences to predict www surfing', *USENIX Annual Technical Conference* pp. 139–150.

Pons, A. P. (2006), 'Object prefetching using semantic links', *The DATA BASE for Advances in Information Systems* **37**(1), 97–109.

Rigou, M., Sirmakesses, S. & Tzimas, G. (2006), 'A method for personalized clustering in data intensive web applications', *APS'06, Denmark* pp. 35–40.

Sarukkai, R. (2000), 'Link prediction and path analysis using markov chains', *9th International WWW Conference, Amsterdam* pp. 377–386.

Spiliopoulou, M., Faulstich, L. C. & Winkler, K. (1999), 'A data miner analysing the navigational behaviour of web users', *Workshop on Machine Learning in User Modelling of the ACAI'99, Greece* .

Srivastava, J., Cooley, R., Deshpande, M. & Tan, P. (2000), 'Web usage mining: Discovery and applications of usage patterns from web data.', *SIGDD Explorations* **1**(2), 12–23.

Strehl, A., Ghosh, J. & Mooney, R. J. (2000), 'Impact of similarity measures on web-page clustering', *AI for Web Search* pp. 58–64.

Wang, Q., Makaroff, D. J. & Edwards, H. K. (2004), 'Characterizing customer groups for an e-commerce website', *EC'04, USA* pp. 218–227.

Yang, Q., Li, T. & Wang, K. (2004), 'Building association-rule based sequential classifiers for web-document prediction', *Journal of Data Mining and Knowledge Discovery* **8**.

Yong, W., Zhanhuai, L. & Yang, Z. (2005), 'Mining sequential association-rule for improving web document prediction', *ICCIMA'05* pp. 146–151.

Zhao, Q., Bhomick, S. S. & Gruenwald, L. (2005), 'Wam miner: In the search of web access motifs from historical web log data', *CIKM'05, Germany* pp. 421–428.

Zhong, S. & Ghosh, J. (2003), 'A unified framework for model-based clustering', *Machine Learning Research* **4**, 1001–1037.

Zhu, J., Hong, J. & Hughes, J. G. (2002), 'Using markov models for web site link prediction', *HT'02, USA* pp. 169–170.

An efficient hash-based algorithm for minimal k-anonymity

Xiaoxun Sun **Min Li** **Hua Wang** **Ashley Plank**

Department of Mathematics & Computing
University of Southern Queensland
Toowoomba, Queensland 4350, Australia
Email: {sunx, limin, wang, plank}@usq.edu.au

Abstract

A number of organizations publish microdata for purposes such as public health and demographic research. Although attributes of microdata that clearly identify individuals, such as name and medical care card number, are generally removed, these databases can sometimes be joined with other public databases on attributes such as Zip code, Gender and Age to re-identify individuals who were supposed to remain anonymous. "Linking" attacks are made easier by the availability of other complementary databases over the Internet.

k-anonymity is a technique that prevents "linking" attacks by generalizing and/or suppressing portions of the released microdata so that no individual can be uniquely distinguished from a group of size k. In this paper, we investigate a practical model of k-anonymity, called full-domain generalization. We examine the issue of computing minimal k-anonymous table based on the definition of minimality described by Samarati. We introduce the hash-based technique previously used in mining associate rules and present an efficient hash-based algorithm to find the minimal k-anonymous table, which improves the previous binary search algorithm first proposed by Samarati.

Keywords: microdata release, hash-based algorithm, k-anonymity.

1 Introduction

Several microdata[1] disclosure protection techniques have been developed in the context of statistical database, such as scrambling and swapping values and adding noise to the data while maintaining an overall statistical integrity of the result (Adam & Wortman 1989, Willenborg & DeWaal 1996). However, many applications require release and explicit management of microdata while maintaining truthful information within each tuple. This 'data quality' requirement makes inappropriate those techniques that disturb data and therefore, although preserving statistical properties, compromise the correctness of the single pieces of information. Among the techniques proposed for providing anonymity in the release of microdata (Federal Committee on Statistical Methodology 1994) we focus on two techniques in particular: generalization and suppression (in the Statistics literature, this approach is often called *recording*), which unlike other existing techniques, such as scrambling or swapping, preserve the truthfulness of the information.

k-anonymity is a technique that prevents joining attacks by generalizing and/or suppressing portions of the released microdata so that no individual can be uniquely distinguished from a group of size k. There are a number of models for producing an anonymous table. One class of models, called *global-recoding* (Willenborg & DeWaal 2001), map the values in the domains of quasi-identifier attributes (defined in Section 2) to other values. This paper is primarily concerned with a specific global-recoding model, called *full-domain generalization*. Full-domain generalization was proposed by Samarati and Sweeney (Samarati & Sweeney 1998, Samarati 2001) and maps the entire domain of each quasi-identifier attribute in a table to a more general domain in its domain generalization hierarchy. This scheme guarantees that all values of a particular attribute in the anonymous table belong to the same domain.

For any anonymity mechanism, it is desirable to define some notion of minimality. Intuitively, a k-anonymous table should not generalize, suppress, or distort the data more than is necessary to achieve such k-anonymity. Indeed, there are a number of ways to define minimality. One notion of minimality is defined as to generalize or suppress the minimum number of attribute values in order to satisfy a given k-anonymity requirement. Such a problem is NP-hard (Aggarwal et al. 2005, Meyerson & Williams 2004). As to our model, the notion of minimal full-domain generalization was defined in (Samarati & Sweeney 1998, Samarati 2001) using the distance vector of the domain generalization. Informally, this definition says that a full-domain generalized private table (PT) is minimal if PT is k-anonymous, and the height of the resulting generalization is less than or equal to that of any other k-anonymous full-domain generalization.

In this paper, we focus on this specific global-recoding model of k-anonymity. Our objective is to find the minimal k-anonymous generalization (table) under the definition of minimality defined by Samarati (Samarati 2001). By introducing the hash-based technique, we provide a new approach to generate minimal k-anonymous tables that not only improves the search algorithm proposed by Samarati (Samarati 2001) but is also useful for computing other optimal criteria for k-anonymity.

The remainder of this paper is organized as follows. In Section 2, we introduce some notions of k-anonymous table. In Section 3, we introduce our hash technique used in this paper. In Section 4, we introduce the generalization relationship and the definition of minimal k-anonymous table. Our core hash-based

[1]The term "microdata" refers to data published in raw, non-aggregated form (Willenborg & DeWaal 2001).

Gender	Age	Zip	Other Attributes	Diseases
Male	25	4370	· · · · · ·	Hypertension
Male	25	4370	· · · · · ·	Hypertension
Male	22	4352	· · · · · ·	Depression
Female	28	4373	· · · · · ·	Chest Pain
Female	28	4373	· · · · · ·	Obesity
Female	34	4350	· · · · · ·	Flu

Table 1: Released microdata

Bucket	0	1	2	3
Contents	(22,4352)	(25, 4370) (25, 4370)	(34, 4350)	(28, 4373) (28, 4373)

Table 2: Hashed table 1 with QI = {Age, Zip}

Bucket	0	1	2	3
COUNT	1	2	1	2
Contents	(22,4352)	(25, 4370) (25, 4370)	(34, 4350)	(28, 4373) (28, 4373)

Table 3: Hash table with COUNT

algorithm and comparisons with previous algorithm discussed in Section 5. Related work is discussed in Section 6. Conclusion and future work are drawn in Section 7.

2 k-anonymous private table

The concept of k-anonymity (Samarati & Sweeney 1998) tries to capture one of the main requirements that has been followed by the statistical community and by agencies releasing data on the private table (PT). According to the requirement, the released data should be indistinguishably related to no less than a certain number of respondents. The set of attributes included in the private table, also externally available and therefore exploitable for linking, is called *quasi-identifier* (QI). The requirement just stated is then translated into the k-anonymity requirement below, which states that every tuple released cannot be related to fewer than k respondents.

Definition 1 (k-anonymous requirement): Each release of data must be such that every combination of values of quasi-identifiers can be indistinctly matched to at least k respondents.

Since it seems impossible or highly impractical to make assumptions on the datasets available for linking to external attackers or curious data recipients, essentially k-anonymity takes a safe approach requiring that the respondents should be indistinguishable (within a given set) with respect to the set of attributes in the released table. To guarantee the k-anonymity requirement, k-anonymity requires each value of a *quasi-identifier* in the released table to have at least k occurrences. Formally, we have the following definition.

Definition 2 (k-anonymity): Let $PT(A_1, \cdots, A_m)$ be a private table and QI be a *quasi-identifier* associated with it. PT is said to satisfy k-anonymity with respect to QI if and only if each sequence of values in $PT[QI]$ appears at least with k occurrences in $PT[QI]$[2].

If a set of attributes of external tables appears in the *quasi-identifier* associated with the private table (PT) and the table satisfies k-anonymity, then the combination of the released data with the external data will never allow the recipient to associate each released tuple with less than k respondents. For instance, consider the released microdata in Table 1 with *quasi-identifier* QI = {Gender, Age, Zip}, we see that the table satisfies k-anonymous with $k = 1$ only since there exists single occurrence of values over the considered QI (e.g., the single occurrence of "Male, 22 and 4352").

[2] $PT[QI]$ denotes the projection, maintaining duplicate tuples, of attributes QI in PT.

3 Hash table

A *hash table* is a data structure that will increase the search efficiency from $O(log(n))$ (binary search) to $O(1)$ (constant time) (Cormen et al. 2001). A *hash table* is made up of two parts: an array (the actual table where the data to be searched is stored) and a mapping function, known as a *hash function*. The *hash function* is a mapping from the input data space to the integer space that defines the indices of the array (bucket). In other words, the hash function provides a way for assigning numbers to the input data such that the data can then be stored at the array (bucket) with the index corresponding to the assigned number. For example, the data in Table 1 are mapped into buckets labeled 0, 1, 2, 3 in Table 2. The data in the bucket with the same assigned number is called a *hash equivalence class*. Depending on the different problems, we could choose different hash functions to classify our input data as we need. For instance, consider quasi-identifier QI = {Age, Zip} in Table 1. We hash them into different buckets with the function $((Age - 20) + (Zip - 4350)) \ mod \ 4$ (see Table 2).

From Table 2 we see that two identical data (25, 4350) and (28, 4353) in the quasi-identifier fall into two different *hash equivalence classes*. Further, if we add a row (labeled COUNT) to record the number of contents in the corresponding bucket (see Table 3), we can easily determine whether or not the table satisfies the k-anonymity requirement. For instance, according to the row COUNT in Table 3, Table 1 only satisfies k-anonymity with $k = 1$.

This hash-based technique is not new in data mining. In (Park et al. 1995), the authors used this technique to present an efficient hash-based algorithm for mining association rules which improves previous well-known *A priori* algorithm. In this paper, we integrate this technique into computation of minimal k-anonymous table. By using such a technique, we can reduce the number of potential sets that need to be checked whether they are k-anonymous during binary search and thus improve the time complexity in (Samarati 2001)

Concerning the efficiency of hash table and binary search, we note the following. (1) Hash table has a faster average lookup time $O(1)$ (Cormen et al. 2001) [3] than the binary search algorithm $O(log(n))$. Hash

[3] Note that the worst case in hash tables happens when every data element are hashed to the same value due to some bad luck in choosing the hash function and bad programming. In that case, to do a lookup, we would really be doing a straight linear search on a linked list, which means that our search operation is back to being $O(n)$. The worst case search time for a hash table is $O(n)$. However, the probability of that happening is so small that, while the worst case search time is $O(n)$, both the best and average cases are $O(1)$.

table shines in very large arrays, where $O(1)$ performance is important. (2) Building a hash table requires a reasonable hash function, which sometimes can be difficult to write well, while binary search requires a total ordering on the input data. On the other hand, with hash tables the data may be only partially ordered.

4 Data generalization

4.1 Generalization relationship

Among the techniques proposed for providing anonymity in the release of microdata, the k-anonymity proposal focuses on two techniques in particular: generalization and suppression, which unlike other existing techniques, such as scrambling or swapping, preserve the truthfulness of the information.

Generalization consists in substituting the values of a given attribute with more general values. We use $*$ to denote the more general value. For instance, we could generalize two different Zip code 4370 and 4373 to 437$*$. The other technique, referred to as data suppression, removes the part or entire value of attributes from the table. Since suppressing an attribute (i.e., not releasing any of its values) to reach k-anonymity can equivalently be modeled via a generalization of all the attribute values to the most generalized data $*$[4], we consider only data generalization.

The notion of *domain* (i.e., the set of values that an attribute can assume) is extended to capture the generalization process by assuming the existence of a set of *generalized domains*. The set of original domains together with their generalizations is referred to as Dom. Each generalized domain contains generalized values and there exists a mapping between each domain and its generalizations. (For example, Zip codes can be generalized by dropping the least significant digit at each generalization step, Ages can be generalized to an interval, and so on). This mapping is described by means of a *generalization relationship* $\leq_D$. Given two domains D_i and $D_j \in$ Dom, $D_i \leq_D D_j$ states that values in domain D_j are generalizations of values in D_i. The *generalization relationship* $\leq_D$ defines a partial order on the set Dom of domains, and is required to satisfy the following two conditions:

C_1: $\forall D_i, D_j, D_z \in$ Dom:
$D_i \leq_D D_j, D_i \leq_D D_z \Rightarrow D_j \leq_D D_z \vee D_z \leq_D D_j$
C_2: all maximal element of Dom are singleton.

Condition C_1 states that for each domain D_i, the set of domains generalization of D_i is totally ordered and we can think of the whole generalization domain as a chain of nodes, and if there is an edge from D_i to D_j, we call D_j the *direct generalization* of D_i. Note that the *generalization relationship* $\leq_D$ is transitive, and thus, if $D_i \leq D_j$ and $D_j \leq D_k$, then $D_i \leq D_k$. In this case, we call D_k the *implied generalization* of D_i. Condition C_1 implies that each D_i has at most one *direct generalization* domain D_j, thus ensuring determinism in the generalization process. Condition C_2 ensures that all values in each domain can be generalized to a single value. For each domain $D \in$ Dom, the definition of a generalization relationship implies the existence of a totally ordered hierarchy, called the *domain generalization hierarchy*, denoted DGH_D. Pathes in the *domain generalization hierarchy* correspond to *implied generalizations* and edges correspond to *direct generalizations*. For example, consider DGH_{Z_0} in Figure 1. Z_1 is the direct generalization of Z_0 and Z_2 is the implied generalization of Z_0.

[4]Note that this observation holds assuming that attribute suppression removes only the values and not the attribute (column) itself. This assumption is reasonable since removal of the attribute (column) is not needed for k-anonymity.

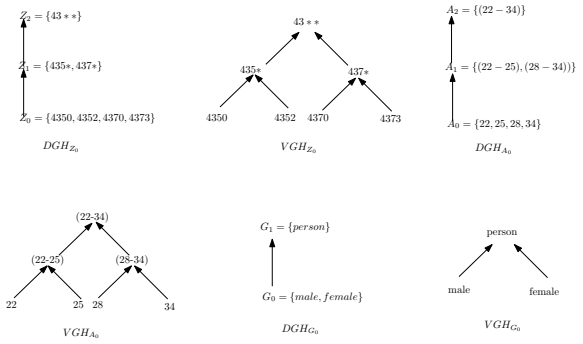


Figure 1: Domain and value generalization hierarchies for Zip, Age and Gender

A *value generalization relationship* denoted $\leq_V$, can also be defined, which associates with each value in domain D_i a unique value in domain D_j. For each domain $D \in$ Dom, the value generalization relationship implies the existence of a *value generalization hierarchy*, denoted VGH_D. It is easy to see that the value generalization hierarchy VGH_D is a tree, where the leaves are the minimal values in D and the root (i.e., the most general value) is the value of the maximum element in DGH_D.

EXAMPLE: Figure 1 illustrates an example of domain and value generalization hierarchies for domains: Z_0, A_0 and G_0. Z_0 represents a subset of the Zip codes in Table 1; A_0 represents Age; and G_0 represents Gender. The generalization relationship specified for Zip codes generalizes a 4-digit Zip code, first to a 3-digit Zip code, and then to a 2-digit Zip code. The attribute Age is first generalized to the interval (22-25) and (28-34), then to the interval (22-34). The Gender hierarchy in the figure is of immediate interpretation.

Since the approach in (Samarati 2001) works on sets of attributes, the generalization relationship and hierarchies are extended to refer to tuples composed of elements of Dom or of their values. Given a domain tuple $DT =< D_1, \cdots, D_n >$ such that $D_i \in$ Dom, $i = 1, \cdots, n$, the domain generalization hierarchy of DT is $DGH_{DT} = DGH_{D_1} \times \cdots \times DGH_{D_n}$, where the Cartesian product is ordered by imposing coordinate-wise order. Since each DGH_{D_i} is totally ordered, DGH_{DT} defines a lattice with DT as its minimal element and the tuple composed of the top of each DGH_{D_i}, $i = 1, \cdots, n$ as its maximal element. Each path from DT to the unique maximal element of DGH_{DT} defines a possible alternative path, called *generalization strategy* for DGH_{DT}, which can be followed when generalizing a quasi-identifier $QI = (A_1, \cdots, A_n)$ of attributes on domains $D_1, \cdots, D_n$. In correspondence with each generalization strategy of a domain tuple, there is a value generalization strategy describing the generalization at the value level. Such a generalization strategy hierarchy is actually a tree structure. The top unique maximal element can be regarded as the root of the tree and the minimal element on the bottom is the leaf of the tree. Let $L[i, j]$ denote the j^{th} data at height i (The bottom data is at the height 0) and $L[i]$ denote the number of data at the height i.

EXAMPLE: Consider domains G_0 (Gender) and Z_0 (Zip code) whose generalization hierarchies are illustrated in Figure 1. Figure 2 illustrates the domain generalization hierarchy of the domain tuple $< G_0, Z_0 >$ together with the corresponding do-

G_0	Z_0
Male	4370
Male	4370
Male	4352
Female	4373
Female	4373
Female	4350

Fig 3. 1: PT

G_0	Z_2
Male	43**
Male	43**
Male	43**
Female	43**
Female	43**
Female	43**

Fig 3. 2: $GT_{[0,2]}$

G_1	Z_1
person	437*
person	437*
person	435*
person	437*
person	437*
person	435*

Fig 3. 3: $GT_{[1,1]}$

G_1	Z_2
person	43**
person	43**
person	43**
person	43**
person	43**
person	43**

Fig 3. 4: $GT_{[1,2]}$

Figure 3: Generalized table for PT

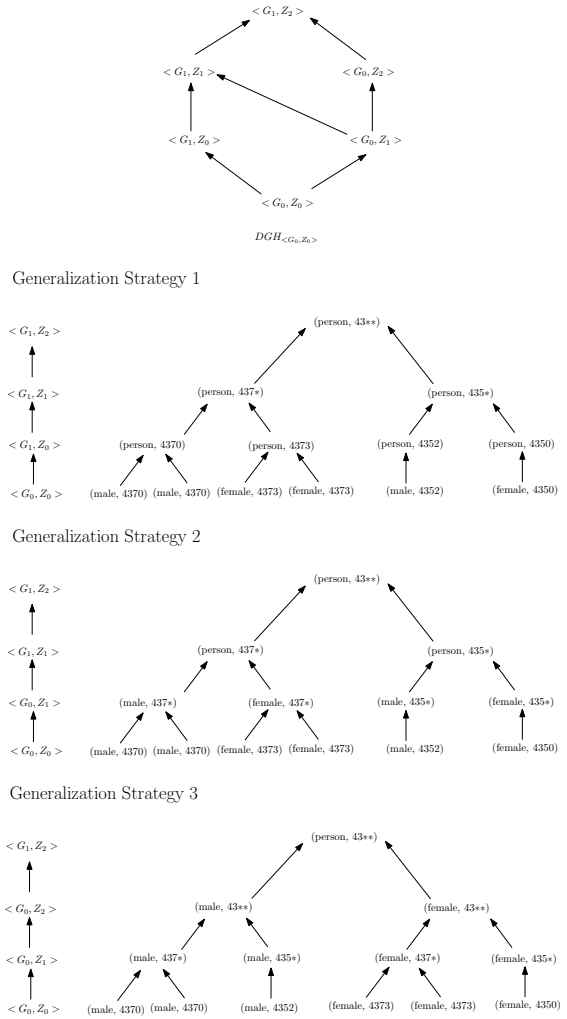


Figure 2: Hierarchy $DGH_{<G_0,Z_0>}$ and corresponding domain and value generalization strategies

main and value generalization strategies. There are three different generalization strategies corresponding to the three paths from the bottom to the top element of lattice $DGH_{<G_0,Z_0>}$. In the generalization strategy 1, $L[0,2]$ is (male, 4370), $L[0] = 6$ and $L[2,2]$ is (person, 435∗), $L[2] = 2$.

4.2 Generalized table and minimal generalization

Given a private table (PT), our approach to provide k-anonymity is to generalize the values stored in the table. Intuitively, attribute values stored in the private table (PT) can be substituted with generalized values upon release. Since multiple values can be mapped to a single generalized value, generalization may decrease the number of distinct tuples, thereby possibly increasing the size of the clusters containing tuples with the same values. We perform generalization at the attribute level. Generalizing an attribute means substituting its values with corresponding values from a more general domain. Generalization at the attribute level ensures that all values of an attribute belong to the same domain. In the following, $dom(A_i, PT)$ denotes the domain of attribute A_i in private table PT.

Definition 3 (Generalized table): Let $PT_i(A_1, \cdots, A_n)$ and $PT_j(A_1, \cdots, A_n)$ be two tables defined in the same set of attributes. PT_j is said to be a generalization of PT_i, written $PT_i \preceq PT_j$, if and only if: (1) $|PT_i| = |PT_j|$; (2) $\forall A_z \in \{A_1, \cdots, A_n\}$: $dom(A_z, PT_i) \leq_D dom(A_z, PT_j)$; and (3) It is possible to define a bijective mapping between PT_i and PT_j that associates each tuple $pt_i \in PT_i$ with a tuple $pt_j \in PT_j$ such that $pt_i[A_z] \leq_V pt_j[A_z]$ for all $A_z \in \{A_1, \cdots, A_n\}$.

EXAMPLE: Consider the private table PT illustrated in Figure 3.1 and the domain and value generalization hierarchies for G_0(Gender) and Z_0(Zip) illustrated in Figure 2. Assume QI = {Gender, Zip} to be a quasi-identifier. The following three tables in Figure 3 are all possible generalized tables for PT. For the clarity, each table reports the domain for each attribute in the table. With respect to k-anonymity, $GT_{[1,1]}$ satisfies k-anonymity for $k = 1, 2$; $GT_{[0,2]}$ satisfies k-anonymity for $k = 1, 2, 3$ and $GT_{[1,2]}$ satisfies k-anonymity for $k = 1, \cdots, 6$.

Given a private table PT, different possible generalizations exist. However, not all generalizations can be considered equally satisfactory. For instance, the trivial generalization bringing each attribute to the highest possible level of generalization provides k-anonymity at the price of a strong generalization of the data. Such extreme generalization is not needed if a table containing more specific values exists which satisfies k-anonymity as well. This concept is captured by the definition of minimal k-anonymity (generalization). To introduce it we first introduce the notion of distance vector. [5]

Definition 4 (Distance vector): Let $PT_i(A_1, \cdots, A_n)$ and $PT_j(A_1, \cdots, A_n)$ be two tables such that $PT_i \preceq PT_j$. The distance vector of PT_j from PT_i is the vector $DV_{i,j} = [d_1, \cdots, d_n]$ where each d_z, $z = 1, \cdots, n$, is the length of the unique path between $D_z = dom(A_z, PT_i)$ and $dom(A_z, PT_j)$ in the domain generalization hierarchy DGH_{D_z}.

EXAMPLE: Consider the private table PT and its generalizations illustrated in Figure 3. The distance vectors between PT and each of its generalized tables is the vector appearing as a subscript of the table. A generalization hierarchy for a domain tuple can be seen as a hierarchy (lattice) on the corresponding distance vectors. Figure 4 illustrates the lattice

[5] In (LeFevre et al. 2005) the star scheme is used for large databases. Here, we use distance vector to define minimal k-anonymity.

Bucket	0	1	2
$Children[i,j]$	0	1	≥ 2
Contents	$L[0,1]$, $L[0,2]$, $L[0,3]$ $L[0,4]$, $L[0,5]$, $L[0,6]$	$L[1,3]$ $L[1,4]$	$L[1,1]$, $L[1,2]$ $L[2,1]$, $L[2,2]$, $L[3,1]$

Table 4: Hash table of Generalization Strategy 1 in Figure 2

Algorithm 1: Finding minimal solution in *k-anonymous class*.

Input: the *k-anonymous class*

1. Sort the data in *k-anonymous class*.
2. Compute the number $n(i)$ of $L[i,j]$ at each height i;
3. If $n(i) \neq L[i]$, discard the all the $L[i,j]$ at the height i.
4. Otherwise, keep them.

Output: The height at which the first data is in the remaining *k-anonymous class*, and generalize the data to this height could obtain the minimal k-anonymous table.

Algorithm 2: Hash-based algorithm for minimal k-anonymity.

Input: Generalization hierarchy and anonymous requirement k

Output: A minimal k-anonymous table.

1. Create a table with $k+1$ column labeling $0, 1, \cdots, k-1, k$. Compute $Children[i,j]$ for each data j at the height i.
2. For $l = 0, 1, \cdots, k-1$ if $Children[i,j] = l$, put $Children[i,j]$ to the bucket labeled l. else put $Children[i,j]$ to the bucket labeled k.
3. Compute the minimal k-anonymous table by Algorithm 1.

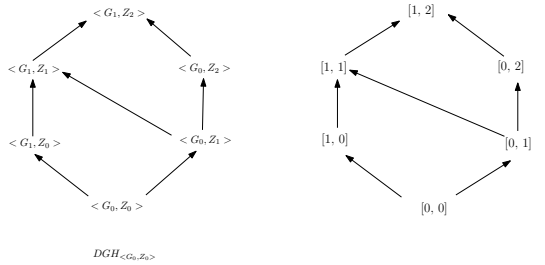


Figure 4: Hierarchy $DGH_{<G_0,Z_0>}$ and corresponding lattice on distance vectors

representing the dominance relationship between the distance vectors corresponding to the possible generalizations of $< G_0, Z_0 >$.

We extend the dominance relationship $\leq_D$ on integers to distance vectors by requiring coordinate-wise ordering as follows. Given two distance vectors $DV = [d_1, \cdots, d_n]$ and $DV' = [d'_1, \cdots, d'_n]$, $DV \leq DV'$ if and only if $d_i \leq d'_i$ for all $i = 1, \cdots, n$. Moreover, $DV < DV'$ if and only if $DV \leq DV'$ and $DV \neq DV'$.

Intuitively, a generalization $PT_i(A_1, \cdots, A_n)$ is minimal k-anonymity (generalization) if and only if there does not exist another generalization $PT_z(A_1, \cdots, A_n)$ satisfying k-anonymity and whose domain tuple is dominated by PT_j in the corresponding lattice of distance vectors. Formally, we can define it as follows:

Definition 5 (Minimal k-anonymity): Let $PT_i(A_1, \cdots, A_n)$ and $PT_j(A_1, \cdots, A_n)$ be two tables such that $PT_i \preceq PT_j$. PT_j is said to be a minimal k-anonymity (generalization) of PT_i if and only if: (1) PT_j satisfies k-anonymity; and (2) $\forall PT_z : PT_i \preceq PT_z$, PT_z satisfies k-anonymity $\Rightarrow \neg(DV_{i,z} \leq DV_{i,j})$.

EXAMPLE: Consider table PT and its generalized tables illustrated in Figure 3. For $k = 2$ two minimal k-anonymous table exist, namely $GT_{[0,2]}$ and $GT_{[1,1]}$. $GT_{[1,2]}$ is not minimal because it is a generation of $GT_{[1,1]}$ and $GT_{[0,2]}$. Also, there is only one minimal k-generalized tables with $k = 3$, which is $GT_{[0,2]}$.

5 Hash-based algorithm

A number of convincing parallels exist between Samarati and Sweeney's generalization framework (Samarati & Sweeney 1998, Samarati 2001) and ideas used in mining association rules (Agrawal & Srikant 1994, Srikant & Agrawal 1995) and the hash-based technique used in (Park et al. 1995). By bringing these techniques to bear on our model of full-domain generalization problem, we develop an efficient hash-based algorithm for computing k-minimal anonymity.

In (Samarati 2001), Samarati describes an algorithm for finding a single minimal k-anonymous full-domain generalization based on the specific definition of minimality outlined in the previous section. The algorithm uses the observation that if no generalization of height h satisfies k-anonymity, then no generalization of height $h' < h$ will satisfy k-anonymity. For this reason, the algorithm performs a binary search on the height value. If the maximum height in the generalization lattice is h, the algorithm begins by checking each generalization at height $\lfloor \frac{h}{2} \rfloor$. If a generalization exists at this height that satisfies k-anonymity, the search proceeds to look at the generalizations of height $\lfloor \frac{h}{4} \rfloor$. Otherwise, generalizations of height $\lfloor \frac{3h}{4} \rfloor$ are searched, and so forth. This algorithm is proven to find a single minimal k-anonymous table.

We integrate the hash technique into the algorithm and develop a more efficient algorithm based on our definition of minimality (Definition 5). A drawback of Samarati's algorithm is that for arbitrary definitions of minimality this binary search algorithm is not always guaranteed to find the minimal k-anonymity table. We conjecture that the hash technique used in this paper might be suitable for the further improvement of algorithms based on other optimal criteria for k-anonymity.

Let the domain generalization hierarchy be DGH_{DT}, where DT is the tuples of the domains of the quasi-identifier. Assume that the top generalization data with the highest height in DGH_{DT} satisfies the required k-anonymity. The idea of the algorithm is to hash the data in DGH_{DT} to a different *hash equivalence class*. Under our definition of the minimality, the hash function that we choose should hash all generalizations with height $h > 0$ in DGH_{DT} that satisfies k-anonymity to the same *hash equivalence class*, which is called the *k-anonymous class*. (The bucket labeled 2 in Table 4). The hash-based algorithm consists of two main steps. At the first stage, the data that satisfies k-anonymity are hashed into the *k-anonymous class*. The second step is to use Algorithm 1 to find the minimal k-anonymous table in the *k-anonymous class*.

Algorithm 1 illustrate how to find the minimal k-anonymous table in *k-anonymous class*. Consider Table 1 and its Generalization Strategy 1 in Figure 2. Generalized data $L[1,1]$, $L[1,2]$, $L[2,1]$, $L[2,2]$ and $L[3,1]$ are hashed into the *k-anonymous class*. We sort the data in *k-anonymous class* as $\{L[1,1], L[1,2], L[2,1], L[2,2], L[3,1]\}$. Since $L[1] = 4$ and the number of data at the height 1 in *k-anonymous class* is 2. According to Step 3 in Al-

gorithm 1, we delete $L[1,1]$ and $L[1,2]$ from *k-anonymous class*. At last, the output height is 2, and we could generalize the table to this height so that it satisfies 2-anonymity with quasi-identifier QI = {Gender, Zip}.

Next, we illustrate how to hash the generalization data in DGH_{DT} to the *k-anonymous class*. Denote $Children[i,j]$ the number of children that the j^{th} data at the height i have. For example, in Generalization Strategy 1 in Figure 2, $Children[1,3] = 1$ and $Children[2,1] = 4$. Suppose we have the requirement of k-anonymity. The desired hash table contains $k+1$ buckets, labeled as $0, 1, 2, \cdots, k-1, k$, the labeled number $0, 1, \cdots, k-1$ denotes the value of $Children[i,j]$ in DGH_{DT} and the k^{th} bucket has the data whose $Children[i,j] \geq k$. Note that the bucket labeled k is actually the *k-anonymous class*. We could see the following Table 4 as an example(where $k = 2$). All the potential generalization data satisfying 2-anonymity are classified into the third bucket, which consists of the *k-anonymous class*.

Algorithm 2 is our hash-based algorithm. Compared to Samarati's binary search algorithm, Algorithm 2 finds the minimal k-anonymous table in the *k-anonymous class*, which is smaller than the potential sets that need to be checked in Samarati's algorithm. Because of the hash technique we used in Algorithm 2, the search complexity is reduced from $O(log(n))$ (binary search) to $O(1)$ (Cormen et al. 2001).

6 Related work

Protecting anonymity when publishing microdata has long been recognized as a problem (Willenborg & De-Waal 2001), and there has been much recent work on computing k-anonymity for this purpose. The μ-Argus system (Hundepool & Willenborg 1996) was implemented to anonymize microdata but considered attribute combinations of only a limited size, so the results were not always guaranteed to be k-anonymous.

In recent years, numerous algorithms have been proposed for implementing k-anonymity via generalization and suppression. The framework was originally defined by Samarati and Sweeney (Samarati & Sweeney 1998). Sweeney proposed a greedy heuristic algorithm for full-domain generalization ("Datafly") (Sweeney 2002). Although the resulting generalization is guaranteed to be k-anonymous, there are no minimality guarantees. Samarati proposed the binary search algorithm for discovering a single minimal full-domain generalization that is described in Section 5. LeFevre et al. described an efficient search algorithm called *Incognito*, for anonymous full-domain generalization (LeFevre et al. 2005).

Cost metrics intended to quantify loss of information due to generalization were described in (Iyengar 2002). Given such a cost metric, Iyengar (Iyengar 2002) developed a genetic algorithm and Winkler (Winkler 2002) described a stochastic algorithm based on simulated annealing to find locally minimal anonymous table. Recently, top-down (Fung et al. 2005) and bottom-up (Wang, Yu & Chakraborty 2004) greedy heuristic algorithms were proposed to produce anonymous data.

Bayardo and Agrawal (Bayardo & Agrawal 2005) described a set enumeration approach to find an optimal anonymous table according to a given cost metric. Subsequent work shows that optimal anonymity under this model may not be as good as anonymity produced with a multi- dimension variation (LeFevre et al. 2005). Finally, Meyerson and Williams (Meyerson & Williams 2004) and Aggarwal et al. (Aggarwal et al. 2005) proved the optimal k-anonymity is NP-hard (based on the number of cells and number of attributes that are generalized and suppressed) and describe approximation algorithms for optimal k-anonymity.

In addition to generalization and suppression, related techniques based on clustering have also been proposed in the literature. Microaggregation first clusters data into (ideally homogeneous) groups of required minimal occupancy and then publishes the centroid of each group (Domingo-Ferrer & Mateo-Sanz 2002). Similarly, Aggarwal et al. propose clustering data into groups of at least size k and then publish various summary statistics for each cluster (Aggarwal et al. 2006).

7 Conclusion and future work

In this paper, we focus on a specific global-recoding model of k-anonymity. Our objective is to find the minimal k-anonymous generalization (table) under the definition of minimality defined by Samarati (Samarati 2001). By introducing the hash-based technique, we provide a new approach to generate minimal k-anonymous table, which not only improves previous search algorithm proposed by Samarati (Samarati 2001), but might be useful for computing other optimal criteria solution for k-anonymity.

In future work, we conjecture this hash-based technique might be suitable for further improvement of the *Incognito* (LeFevre et al. 2005) for full-domain generalization, since it might significantly reduce the number of 2-attribute candidate sets. The technique might also apply in multilevel generalization. For many applications, it is difficult to find required k-anonymity tables at low or primitive levels of the generalization hierarchy due to the sparsity of data in multidimensional space. k-anonymous table generated at very high levels in the generalization hierarchy might be suitable for some attributes, however, to some extent, it may be too generalized in some attributes. Therefore, data mining system should provide capability to generate k-anonymous tables at multiple levels of the generalization hierarchy and traverse easily among different generalization levels. The hash-based technique may provide a new point of view and a more efficient way to make multilevel k-anonymous tables.

References

Adam, N. R. and Wortman, J. C. Security-Control Methods for Statistical Databases: A Comparative Study, ACM Computing Surveys, vol. 21, no. 4, pp. 515-556, 1989.

Aggarwal, G., Feder, T., Kenthapadi, K., Motwani, R., Panigrahy, R., Thomas, D. and Zhu, A. Anonymizing tables. In Proc. of the 10th International Conference on Database Theory (ICDT05), pp. 246-258, Edinburgh, Scotland.

Aggarwal, G., Feder, T., Kenthapadi, K., Panigrahy, R., Thomas, D. and Zhu, A. Achieving anonymity via clustering in a metric space. In Proceedings of the 25th ACM SIGACTSIGMOD- SIGART Symposium on Principles of Database Systems (PODS), 2006.

Agrawal, R and Srikant, R. Fast algorithms for mining association rules. In Proc. of the 20th Int'l Conference on Very Large Databases, August 1994.

Bayardo, R. and Agrawal, R. Data privacy through optimal k-anonymity. In Proceedings of the 21st

International Conference on Data Engineering (ICDE), 2005.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. Introduction to Algorithms, second edition, MIT Press and McGraw-Hill. ISBN 0-262-53196-8.

Domingo-Ferrer, J and Mateo-Sanz, J. M. Practical data-oriented microaggregation for statistical disclosure control. IEEE Transactions on Knowledge and Data Engineering, 4(1), 2002.

Federal Committee on Statistical Methodology, Statistical Policy Working Paper 22, Report on Statistical Disclosure Limitation Methodology, May 1994.

Fung B, Wang K and Yu P. Top-down specialization for information and privacy preservation. *In* Proc. of the 21st International Conference on Data Engineering (ICDE'05), Tokyo, Japan.

Hundepool, A. and Willenborg, L. μ and τ-ARGUS: Software for statistical disclosure control. In Proc. of the Third International Seminar on Statistical Confidentiality, 1996.

Iyengar V. Transforming data to satisfy privacy constraints. *In* Proc. of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 279-288, Edmonton, Alberta, Canada, 2002.

LeFevre K, DeWitt DJ, Ramakrishnan R. Incognito: Efficient fulldomain k-anonymity. *In* Proc. of the 24th ACM SIGMOD International Conference on Management of Data, pp. 49-60, Baltimore, Maryland, USA, 2005.

LeFevre, K., DeWitt, D. and Ramakrishnan, R. Multidimensional k-anonymity. Technical Report 1521, University of Wisconsin, 2005.

Meyerson A and Williams R. On the complexity of optimal k-anonymity. *In* Proc. of the 23rd ACM-SIGMOD-SIGACT-SIGART Symposium on the Principles of Database Systems, pp. 223-228, Paris, France, 2004.

Park, J. S., Chen, M. S. and Yu, P. S. An Effective Hash-Based Algorithm for Mining Association Rules. Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data. PP. 175-186, 1995.

Samarati, P and Sweeney, L. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Technical Report SRI-CSL-98-04, SRI Computer Science Laboratory, 1998.

Samarati P. Protecting respondents' identities in microdata release. *IEEE Transactions on Knowledge and Data Engineering*, 13(6):1010-1027. 2001

Samarati P, Sweeney L. Generalizing Data to Provide Anonymity when Disclosing Information (Abstract). *In* Proc. of ACM Symposium on Principles of Database Systems, pp. 188, 1998.

Srikant, R and Agrawal, R. Mining generalized association rules. In Proc. of the 21st Int'l Conference on Very Large Databases, August 1995.

Sweeney L. Achieving k-anonynity privacy protection using generalization and suppression. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 2002, 10(5):571-588.

Wang, K., Yu, P. S and Chakraborty, S. Bottom-up generalization: A data mining solution to privacy protection. In Proceedings of the 4th IEEE International Conference on Data Mining (ICDM), 2004.

Willenborg, L and DeWaal, T. Statistical Disclosure Control in Practice. Springer-Verlag, 1996.

Willenborg, L and DeWaal, T. Elements of Statistical Disclosure Control. Springer Verlag Lecture Notes in Statistics, 2000.

Winkler, W. Using simulated annealing for k-anonymity. Research Report 2002-07, US Census Bureau Statistical Research Division, 2002.

JWS: A Flexible Web Service

Andrew Cho, Paresh Deva, Ewan Tempero
Department of Computer Science
University of Auckland
Auckland, New Zealand
ewan@cs.auckland.ac.nz

Abstract

Web services have been proposed as means to provide more convenient access to computation services. An issue that still must be dealt with is what to do if there is no web service with the desired functionality. Deploying a new web service requires expertise in the relevant technologies as well as access to a web services server. In this paper we present the Java Web Service, a web service that allows the provision of almost arbitrary functionality by means of uploading the functionality as a *plug-in* at run-time. Plug-ins can also be combined through a simple scripting mechanism.

Keywords: Flexible computation service, Web services, Distributed systems.

1 Introduction

The ability to access computation across a network or to distribute computation around a network has been a goal of distributed systems research for many years. Each new generation of distributed systems technology removes one more barrier to providing such an ability. The most recent step has been the introduction of *web services*, which provide network access to to software systems in an *interoperable* manner (Booth, D. et al. 2004), meaning that use of the systems is language and platform independent.

An issue that still remains with web services is that the services they offer are under the control of whoever controls the servers. If the required functionality does not exist as a web service, then there are few options. Those that need new functionality can create it themselves but if they want to make it available to others then they face starting and managing their own server. For those who would rather not take this route, we propose creating a web service that is *flexible* — the functionality it provides can be added to at runtime by any user without affecting existing users. In this paper, we describe the *Java Web Service* (JWS), a web service that can be configured at runtime to allow almost arbitrary functionality.

Web services are claimed to provide several advantages over other distributed computing technologies, including:

- Interoperability — Requests and responses are encoded in neutral formats, such as XML. This means there is no requirement that client and server have to agree in advance on programming language, operating system, or hardware, in order to communication. For example, a Java web service client running on OS X could communicate with a web service provider implemented in C# running on windows.
- Reuse — The interoperability aspect of web services allows an existing module to be wrapped by a web service implementation and have its functionality made available to clients on different software platforms. This avoids the need to duplicate functionality for each platform.
- Open standards — Web services use open non-proprietary standards such as XML and HTTP. This ensures no one company has control over web services. This has helped the popularity of web services.

We wish to retain these advantages as much as possible, while at the same time adding:

- Flexibility — Provide a general computation service by supporting any functionality. This flexibility should not come at the cost of down time. Many clients may be using the service through the Internet at any time and so restarting the service with new functionality is not practical.
- Security — The service should not cause harm to the server that it is running on, such as the deletion of files.

The basic approach we take is to develop a web service with a *plug-in* architecture, in a manner similar to applications such as Eclipse (Gamma & Beck 2004). New functionality can be uploaded to the web services server at run-time, and plug-ins can be combined through a simple scripting mechanism.

The remainder of the paper is organised as follows. The next section provides the background on web services necessary for understanding the rest of the paper. Section 3 describes JWS, with details on the design and implementation in section 4. Section 5 gives an example use of JWS. In section 6 we give an evaluation of the success of JWS, and then discuss related work in section 7. Finally, we give our conclusions and discuss possible future work in section 8.

2 Web Services

Web services are software systems that can be used over a network in an interoperable manner (Booth, D. et al. 2004). They can be used by any program that can process eXtensible Markup Language (XML) and access a network. The simplest form of web services involves two parties: a web service *provider* (server) and a web service *requester* (client). The web service provider makes available some functionality that it performs on behalf of the web service clients. The client, of which there could be many, consumes the

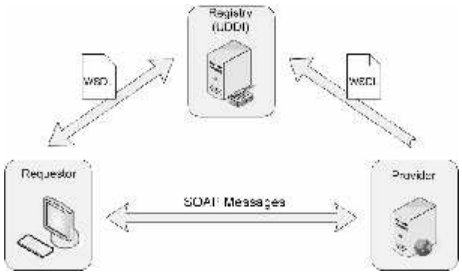


Figure 1: Web Service Architecture Diagram

web service and calls the provider's functionality. The web services concept can also involve a third party, a web services *registry*. Clients may browse this registry for web service providers with specific functionality and bind to them dynamically. Figure 1 illustrates this case.

The provider is responsible for publishing its functionality by implementing a service interface. Providers are described through a service description file. This file is written in the *Web Service Description Language* (WSDL), which is a language for describing web services based on XML. The service description file describes the web service's functionality, such as the available methods, arguments and return types, and the service's network location, and is published to the registry (Booth, D. et al. 2004).

A web service client may not know exactly which web service provider it will interact with. Therefore the client must "discover" a web service provider. Web services can be discovered either manually or automatically. In manual discovery, the client's developer may hard-code the client to use a particular web service. In automatic discovery, web service providers publish their WSDL documents to a registry service, such as Universal Description Discovery and Integration (UDDI). Clients then use this registry service to search for suitable web service providers, obtain their WSDL documents, and bind to them dynamically.

After binding, the requester and provider communicate by sending *Simple Object Access Protocol* (SOAP) messages to each other. SOAP encodes the parameters and results of a service invocation into XML. These messages are transported over a communication protocol such as HTTP, SMTP, FTP, IIOP, or proprietary protocols.

3 The Java Web Service

The JWS provides a flexible computation service that clients can invoke over a network. It provides flexibility in two ways: the JWS can compile and execute almost any Java source code, and JWS has the ability to have its functionality extended by uploading *plug-ins*. These two features combine to allow the JWS to support almost arbitrary functionality.

Figure 2 provides an overview of the JWS. The JWS has a minimalist core of a web service interface, script manager, and plug-in manager. The web service interface receives requests from web service clients and forwards them to the appropriate component. The plug-in manager is responsible for loading, configuring, and invoking plug-ins. The script manager invokes scripts. Plug-ins and scripts are described in more detail in sections 3.2 and 3.3 respectively.

By themselves, these core components provide very little functionality for clients. All of the useful functionality are implemented as plug-ins. Plug-ins provide the first mechanism for the JWS's flexibility in that new functionality can be uploaded to the

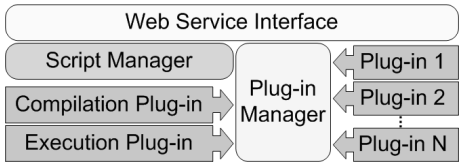


Figure 2: Overview of the Java Web Service

```
<?xml version="1.0" encoding="UTF-8"?>
<ExecutionResult ExitValue="0"
         PluginName="ExecutionPlugin">
  <ExecutionSuccessful>
    true
  </ExecutionSuccessful>
  <ExitValue>
    0
  </ExitValue>
  <Filename>
    TestExecSampleClass
  </Filename>
  <StandardOutput>
    Hello World!
  </StandardOutput>
</ExecutionResult>
```

Figure 3: Example document describing the result from the `execute` method applied to `TestExecSampleClass`
.

JWS as a plug-in. They are integrated dynamically into the JWS while it is running and this integration causes no down time.

The compilation and execution plug-ins are responsible for compiling and executing Java source code respectively. They provide the second mechanism for the JWS's flexibility in that they can be used to execute almost any Java source code. These plug-ins are bundled with the JWS and are always available.

3.1 JWS Service Methods

The JWS contains several different service methods that can be invoked by its clients. These are:

`compile` This method takes one argument that is a string. The string represents the code that the client requires to be compiled. The compilation plug-in is called from this method and the result of this call is a string (an XML document) describing the result of the compilation that is sent back to the client.

`execute` This method takes one argument that is the name of a Java class file and calls the execution plug-in to execute this file. There is a form of this method that takes an integer as a second argument. This argument specifies the amount of the time that the execution is allowed to continue for before the execution process is killed. In the first form of this method, the time is defaults to ten seconds. As with `compile`, an XML document is returned describing the result of the execution. An example is given in figure 3.

`uploadPlugin` This is the method that installs a plug-in provided by the client. Its argument is an array of byte codes. This method converts the data into a file and saves this file in the plug-in repository directory (specified at start-up). A boolean is returned indicate whether or not the upload was successful.

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin name="EchoPlugin"
        class="echoplugin.EchoPlugin">
  <pluginDescription>
    This plug-in is just for testing
  </pluginDescription>
  <method>
    <name>
      echo
    </name>
    <argument>
      java.lang.String
    </argument>
    <description>
      Echoes back the given String
    </description>
  </method>
</plugin>
```

Figure 4: An example of a plug-in manifest file

`getListOfPlugins` This method is used to send information to the client about what plug-ins currently exist in the plug-in directory. It returns an XML document containing the names and descriptions of all of plug-ins. It also contains the names, descriptions and arguments of all of the methods of the plug-ins.

`runScript` The runScript method is used to run a script submitted by the client. It accepts two arguments: script and input. The script is a string and is of the format specified in section 3.3. The input is a string that represents the argument to be used for the first plug-in as specified in the script. The Script Manager executes the script and returns an XML document describing the result of the execution.

3.2 Plug-ins

A plug-in is a small program that provides certain functionality. It can be uploaded to the JWS in order to add new functionality. The JWS's plug-in architecture resembles a simplified version of the Eclipse plug-in architecture (Gamma & Beck 2004). Eclipse is also an application that has a minimalist core of functionality that is extended by plug-ins. As with Eclipse, plug-ins for JWS are packaged within a Java Archive (JAR) file. This archive contains the plug-in's class files and an XML manifest file. The manifest tells the JWS the name of the plug-in, the class to instantiate, and the methods the plug-in has to offer. An example of a manifest file is shown in Figure 4.

For the example in figure 4, the name of the plug-in is "EchoPlugin" and its implementation class is "echoPlugin.EchoPlugin." It only defines one method, "echo", which accepts a `java.lang.String` object as an argument.

The plug-in's implementation class must implement an interface specific to JWS called `IPlugin` so as to be recognised by the JWS as a plug-in. This interface defines two methods: `init()` and `shutdown()`. These methods are called just after instantiation and before removal respectively. They give the plug-in the opportunity to configure and release resources.

3.3 Scripts

Clients submit scripts to the JWS that define which plug-ins should be invoked. Ideally the JWS would publish any methods that plug-ins make available in its WSDL document. However if a new plug-in was uploaded, then the WSDL document would need to change and this would require the web service's clients to re-acquire the latest version of the WSDL and possibly be recompiled. The JWS would also need to be recompiled and re-deployed to Apache Axis, because the JWS's implementation class would need to implement these new methods. As we want to avoid down time, this approach is impractical.

```
NumberPlugin multiplyBy3
EchoPlugin echo
```

Figure 5: An example of a simple plug-in script

```
IF FAIL NumberPlugin multiplyBy3
  EXIT
ELSE
  EchoPlugin echo
ENDIF
```

Figure 6: An example of a plug-in script with conditional behaviour

Scripts allow the WSDL document to remain the same while still enabling new plug-ins to be invoked. As mentioned above, the JWS defines a method, `runScript(Stringscript,Stringinput)`, that executes the given script using the given input, and returns the result of executing the script. A script is essentially a list of plug-ins in the order they are to be executed. Scripts operate in a similar fashion to the UNIX pipe and filter interaction paradigm with the output of one plug-in being passed to the input of the next plug-in.

An example of a script is shown in Figure 5. This simple script uses two plug-ins. The plug-in named `NumberPlugin` has a method called `multiplyBy3` that converts the script's input to a number, multiplies it by three, and returns the result. If the input into this script were "100" then the output after the first line of the script would be "300." This result is passed onto the plug-in named `EchoPlugin`. The echo method of this plug-in concatenates its input string with itself. If this method were executed with an input of "300" the result would be "300300." Therefore the result of executing the example script with an input of "100" is "300300."

Scripts also have the ability to support conditional behaviour based upon whether a plug-in succeeds or not. Figure 6 shows an example of such a script. This script says: if the `NumberPlugin` fails then exit the script, otherwise use the `EchoPlugin`. The `NumberPlugin` will fail if the input into the script can not be converted into a number. Conditional behaviour allows the path of execution to change based upon the outcome of a plug-in.

Plug-ins can tell the JWS whether they have failed in their execution in two ways. Firstly, a plug-in can set an exit value state variable that the JWS will check. This exit value uses the convention that a value of zero means the plug-in was successful and a non-zero value means failure. The second method involves the plug-in returning an XML document. The root node of the document defines the attribute "ExitValue" that contains the exit value described above (similar to that shown in figure 3). The exit value contained within the document will override the exit value contained within the state variable.

The scripting language is very simple, providing

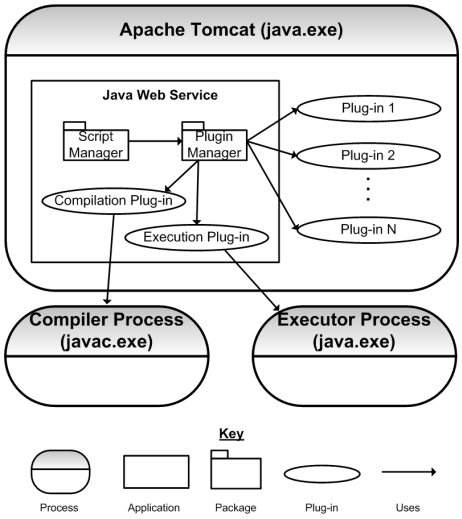


Figure 7: Architecture of the JWS

only the sequencing and conditional behaviour shown here, however we believe that this is sufficient for a wide range of applications.

4 Design and Implementation

4.1 Technologies

Java was chosen because the Java Virtual Machine (JVM) allows the same compiled source code to be run on all operating systems where the JVM is available. This allows development and deployment to become operating system independent.

The web server chosen was Apache Tomcat (Apache 2006*a*) due to its ease of deployment and local expertise. By itself, Apache Tomcat can not host web services because it does not include an implementation of SOAP. Apache Axis (Apache 2006*b*) provides this implementation and also includes several tools to aid web service development and deployment.

From the perspective of the JWS's clients, the technology choices have negligible impact. Web services are interoperable and the implementation details of the service, such as the programming language and choice of server, are hidden away from clients. The only impact of these choices is that plug-ins must be written in Java. Allowing plug-ins to be written in any language was outside the scope of this project but would further improve the flexibility of the JWS.

4.2 Architecture

Figure 7 shows the process structure of the JWS. All of the processes shown reside on the server that hosts the JWS. The main process running on the JWS server is the Apache Tomcat JVM. This process runs the Apache Axis web application (not shown), which in turn runs the JWS. The script manager processes incoming scripts and uses the plug-in manager to invoke plug-ins. The compilation and execution plug-ins are shown to be within the JWS application because they are always available.

To compile Java source code the compilation plug-in writes the source code to a `.java` file and saves it on the file system. Then the `javac` process is started, which compiles the `.java` file to a `.class` file. The execution plug-in then starts the `java` process to execute the `.class` file in its own JVM. The plug-in returns any output that the program sent to either standard output or standard error.

Unlike submitted Java source code, uploaded plug-ins are loaded and executed within the same JVM as the JWS. Submitted Java source code is executed in its own JVM because it can generate output by calling `System.out.println(...)`. This method returns output to the JVM's console and not to the caller. There is only one console per JVM and if multiple programs sent their output to the console it would be difficult to differentiate the output of one program from another. There is a performance cost for this decision as we discuss in section 6.

4.3 Security

The security concerns for the JWS are more prevalent than traditional web services. With traditional web services the service provider is not modified by the client. The JWS is not only modified by uploading plug-ins but arbitrary Java code can also be executed. Both uploaded plug-ins and submitted source code could attempt to cause harm to the server. This includes manipulating the file system (reading, modifying, or deleting files), starting new processes (starting a UNIX shell and executing commands), or opening a network connection allowing back-door access to the server.

To minimise the effect of malicious code harming the server, Java's security manager is enabled for both the Apache Tomcat JVM and the JVM in which submitted Java source code is executed. By default the security manager gives all code none of the permissions specified within the Java Security Architecture framework (Gong 2002). Java security policy files are used to grant certain permissions to certain Java classes. The compilation plug-in is allowed to write to the file system because the compilation process requires writing `.java` and `.class` files to the file system. The execution plug-in is allowed to read these files. To maximise security, all uploaded plug-ins are given no permissions. Uploaded plug-ins can not be trusted because they could be written by anyone.

Security policy files are incapable of preventing infinite loops. An infinite loop is a situation where a program executes continuously inside a loop and never reaches the condition that tells the program to exit the loop. To prevent infinite loops from wasting CPU time on the server, the JWS executes submitted Java code in its own process and kills its process if it goes beyond a certain time limit. This time limit can be modified to the client's demands. More expensive computational tasks will require a longer timeout. However uploaded plug-ins are executed in the same process as the JWS so this approach can not be used for plug-ins that are in an infinite loop. This and other security issues are discussed in section 6.

4.4 Communicating between plug-ins

To achieve complex behaviour, plug-ins may be chained together in a script. For example there may be the need to invoke the execution plug-in after using compilation plug-in. Coordination between plug-ins is difficult because plug-ins could potentially be developed by different developers. If the output of one plug-in does not match the input of the next plug-in then the communication breaks down and the operation will fail. This issue is further complicated by plug-ins that could potentially pass anything (text, files, images) to the next plug-in.

Ideally a plug-in will accept and return a string of text formatted in XML. XML is helpful because it can be used to represent any object and is machine readable, allowing it to be processed dynamically at runtime.

Although passing objects instead of XML would have improved performance (the task of transforming objects to XML and back to objects again would be avoided) it would be a less flexible solution. For a plug-in to pass an object to another plug-in, the receiving plug-in must have an implementation of that object. Therefore there would need to be an agreement between the two plug-ins at design time. This would severely limit the number of plug-ins a given plug-in could interact with.

Plug-ins still need to agree to what XML they expect to receive and produce. This is done by plug-ins providing a Document Type Definition (DTD) file of the XML they produce. This file is machine readable and is a standard way of defining the structure of an XML document.

5 JWS in Action

This section demonstrates how the flexibility of JWS would be used to build an application. The application we developed is called the Online Learning Application (OLA). This application aims to help beginner Java programmers learn the Java syntax. Students go to the OLA's website and are asked to perform Java programming exercises. An example exercise may be to write a for-loop that prints the numbers from one to ten. Students would then write a small piece of Java code that performs this task. The OLA would then compile and execute the student's submission, and compare the result of execution against an expected answer.

Using the OLA as a motivating example for the JWS raises some challenging issues. Novel solutions are required because these issues are not encountered with traditional web service implementations. These issues are outlined below:

- In addition to the JWS acting as an execution engine by executing Java code, it must also be able to compile Java code. Together, compilation and execution provide a flexible mechanism for performing computational tasks.
- Security concerns arise because the JWS could be asked to execute any code that students submit to it. This code could attempt to delete files from the server. Such behaviour must be prevented.
- Plug-ins must be able to be integrated into the JWS despite plug-ins and the JWS being developed by different people. Standardisation is required so that all plug-ins can be integrated into the JWS.
- It is desirable to have plug-ins that can communicate with each other. There must be some agreement on how to pass output from one plug-in to the input of the next plug-in. For example, the compilation plug-in should pass its output to the execution plug-in. The execution plug-in should be able to interpret the compilation plug-in's output and execute any compiled classes.
- Functionality should also be invoked conditionally. For example if the compilation plug-in fails in compiling Java source code, the execution plug-in should not be invoked.
- Some plug-ins require the allocation of machine-dependent resources. For example, the compilation plug-in must have access to a directory to compile its `.class` files to. This directory will be different for every server the plug-in is deployed on.

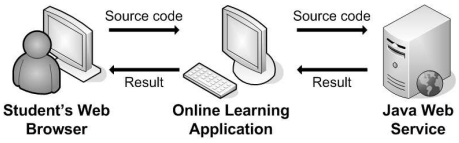


Figure 8: Overview of the Online Learning Application

```
for (int i = 1; i <= 10; i++)
     System.out.println(i);
```

Figure 9: Expected submission for a Java exercise

An overview of the OLA and its relationship with the JWS is shown in Figure 8. The OLA does not implement much functionality. It essentially passes students' submissions to the JWS for compilation and execution, and checks the result against an expected answer. This feedback is then passed back to the student. This demonstration illustrates how applications can be composed quickly by leveraging existing services. Any gaps in functionality can be filled by uploading a plug-in to the JWS.

Students access the OLA through their web browser and complete Java exercises. If the OLA asked the student to write the code that prints the numbers from one to ten, the student would be expected to write something similar to the code shown in Figure 9.

The OLA uses the script shown in Figure 10, with the student's code snippet as the input, to invoke the JWS.

First, the `WrapperPlugin`'s wrap method is invoked. Java requires code to be declared within a class. This plug-in wraps the student's submission with a class declaration and returns the result. For the above example, this plug-in would produce something similar to the class shown in Figure 11.

The name of class is made unique by appending a random number to the end. This minimises the chance that saving the `.java` and `.class` files to the file system will conflict with existing `.java` and `.class` files. These files need to be periodically deleted.

The rest of the script says the code should be compiled with the `CompilationPlugin`. If compilation is successful then `ExecutionPlugin` is used to execute the code. Any output from standard output or standard error is sent back to the OLA. The OLA checks this result against an expected answer and informs the student whether they were correct or not.

If the student's submission does not compile, the script will return the compiler's compilation error messages. These messages are difficult to understand and pose a significant obstacle for beginner programmers (Flowers et al. 2004, Lang 2002). It would be helpful if the OLA had the ability to convert these messages to a more understandable format. This gap in functionality can be filled by developing a plug-in.

```
WrapperPlugin wrap
IF SUCCESS CompilationPlugin compile
    ExecutionPlugin execute
ENDIF
```

Figure 10: Script used by the OLA to invoke the JWS

```
public class Test123456789
  public static
         void main(String[] args)
    for (int i = 1; i <= 10; i++)
      System.out.println(i);
```

Figure 11: Student's code inside a class declaration

```
WrapperPlugin wrap
IF SUCCESS CompilationPlugin compile
    ExecutionPlugin execute
ELSE
    CompilationErrorPlugin transform
ENDIF
```

Figure 12: Script with the CompilationErrorPlugin

`CompilationErrorPlugin` was developed to perform this task. It examines the compiler's error messages, attempts to match them against a catalogue of errors, and replaces them with a more descriptive message. The only change the OLA requires to use this new plug-in is to add an else condition to its script as shown in Figure 12.

This new script says if compilation does not succeed, then the compilation error plug-in should be used to transform the output of the compilation plug-in (compilation errors) into a more understandable format.

The potential for functionality of the OLA is not limited by the JWS. For example a user could decide to extend the OLA to accept a language other than Java by writing a plug-in that would provide this functionality, or if a user feels the `CompilationErrorPlugin` plug-in is insufficient, they can write a better one to be used in its place.

Similarly, the OLA does very simple checking that the student's submission is correct. This checking could be made more sophisticated, in which case it could be incorporated into the OLA application code itself, or made in to a plug-in and uploaded to the JWS server. This plug-in would then be available for anyone else to use.

The OLA requires user code to be executed on the web server, so security becomes an issue. The security policy (mentioned in section 4) for the user-submitted code is specified just before the code is actually run. This is allowable as security policies can be specified when a process is started up for execution, which occurs when the execution plug-in is used. This allows different policies to be specified for different code; so trusted users can use a less stringent policy, which grants more permissions.

For the OLA, a timeout facility is used to ensure that user submitted code that would never complete, such as infinite loops that would continually run but never get any closer to finishing, does not stall the server. After a set amount of time the code would time out (the 'timeout time'). This is implemented utilising the timeOutTime specified in the execution plug-in. For the OLA, the timeout time is set to ten seconds, as only beginner code is expected to be submitted, resulting in quick execution times.

6 Evaluation

The primary goals of this work were to develop a distributed computation system with functionality that is not determined just at deployment time, but allows changes to functionality with no down time, is interoperable, that is, was usable by any client regardless of the programming language, operating system, or hardware of the client, and is secure. By using web services and Java for interoperability, and a plug-in architecture for flexibility, JWS does, in theory, meet these goals.

We have provided evidence of the JWS's flexibility and interoperability in a more practical sense by developing OLA. We face that same problem of any new "enabler" technology. The technology by itself provides no directly observable functionality, so to really prove its usefulness requires devoting significant resources developing many different uses of it. In our case, we chose OLA because it exercises all the features provided by JWS in a single application, and we feel it is representative of a large class of applications of the kind that JWS is intended to support.

In particular, OLA has requirements that motivate the need for flexibility. Had OLA been developed as a standard web service, then its behaviour would be fixed at that time. If, for example, it is determined that the deployed presentation of compilation errors is not informative enough, the users must wait until a better presentation is produced and deployed. With our implementation of OLA users can provide their own plug-in to present such information.

JWS does, however, have its limitations.

- Computational tasks must be specified in Java, either by submitting Java source code or a plug-in. No support is provided for executing source code in other programming languages.
- The JWS is a designed to perform computational tasks and can not be used to display user interfaces and graphics.
- The Java security manager prevents code from causing harm to the server. Any tasks that require operations such as writing to the file system, the creation of new processes, or use of network connections will result in security violations. Currently there is no mechanism that grants permissions to uploaded plug-ins.

As mentioned earlier, the decision to execute submitted Java source code in its own process as opposed to loading it into the same process as JWS has a performance cost. Figure 13 gives an indication as to what this cost is. It shows the cost of performing a compilation, execution in a new process, or execution in the existing process. These timings were done using a Java class that executed an empty for-loop for approximately 16 milliseconds. The tests were run 100 times and the results averaged to minimise random variation. While this is a fairly simplistic test, it does provide the information we are interested in. From the test we can see that executing in a new process takes on average 140 milliseconds longer than executing in the same process. In the context of the various other performance costs associated with web services, we feel this extra time is acceptable.

There are a number of directions that JWS could go. As JWS provides a general computation service, it would be interesting to extend it to provide a distributed *parallel* computation service. To realise the benefits of parallel computing, this would require developing a a plug-in that allows one JWS server to communicate to other JWS servers. This plug-in would need to disseminate the computational task

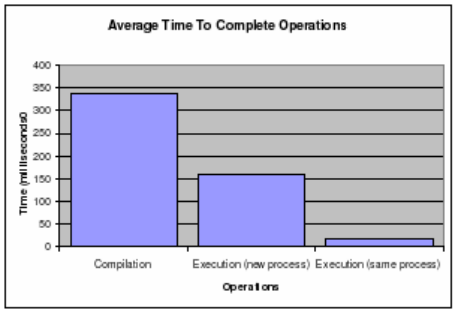


Figure 13: Performance of compilation and execution

amongst the other JWSs and recombine the results. This plug-in would require permission to open network connections. Ideally this plug-in should be bundled with the JWS in a similar manner as the compilation and execution plug-ins.

If parallel computation can be realised the JWS will resemble the Globus (Globus 2006) system. Globus provides the Globus Toolkit, which is used to create grid computing solutions and has become a middleware standard for a number of grid projects. Globus' web services grid resource allocation manager (WS-GRAM) provides dynamic deployment of web services.

The current form of JWS is as secure as the Java security manager is. The security policy we use rules out most security concerns when using JWS. As mentioned in section 4.3, security policies cannot prevent all malicious behaviour from occurring. An uploaded plug-in could execute an infinite loop and the JWS will be blocked while waiting for plug-in to finish. Since uploaded plug-ins are executed in the same process as the JWS itself, killing the process would also kill the JWS, Apache Tomcat, and any other web applications Apache Tomcat was hosting at the time. This issue could be addressed by executing uploaded plug-ins within their own process and killing their process if a timeout is exceeded. This is the same method for terminating infinite loops in submitted Java source code.

Like other web services, the JWS can benefit from the incorporation of the Web Services Security (WSS) policy (Open 2006). This policy aims to provide web services with message integrity and confidentiality. It specifies mechanisms for encrypting messages and authenticating clients and servers.

The WSS policy mentioned above provides a mechanism for authenticating clients but does not provide a mechanism for authenticating uploading plug-ins. It could be possible for an authenticated client to upload an untrustworthy plug-in. This could be addressed by using the JAR signing and verification tool, which is part of the Java Security Architecture (Gong 2002). This tool allows JAR files to be digitally signed so that the JWS can be sure about the origins of the plug-in. This is only part of solution because there must be some mechanism that ensures only trustworthy plug-ins are signed.

Currently the only resource that uploaded plug-ins are allowed to manipulate are the arguments that are passed to it. To allow more powerful plug-ins, such as the parallel computing plug-in, plug-ins must be able to request resources from the JWS. Granting resources is non-trivial because the JWS will not know what resources a plug-in requires until it is requested. In addition, the JWS and plug-ins have no prior agreement, therefore an arbitrary plug-in could request an arbitrary resource. The compilation and execution plug-ins are given a directory to write files to through the plug-ins' constructors. However these plug-ins are special case because they were developed as an integral part of the JWS. Foreign plug-ins can not be passed resources through their constructor because the JWS has no idea what resources are required.

The plug-in could specify the type of resource it requires in its manifest file. For example, the parallel computing plug-in could request a network connection by specifying the value "network connection" within a resources tag element and ideally the JWS will grant the plug-in the ability to create network connections. But there are still issues, such has how the JWS and plug-in developers know that the value "network connection" should be used as opposed to "socket connection," "network," or even "network permission".

Even if the JWS understood what the plug-in was requesting, it still may not know what object to return. For example, if the JWS was asked to provide a directory it may not be clear whether the JWS should return a string that represents a path to the directory, a URL, or a `java.io.File` object.

Prior agreement is required. The JWS could publish a list of resources and the values used to retrieve them as comments in its WSDL document or even a website. However this requires the plug-in developer to know the JWS it will be uploading the plug-in to at design time and modifying the plug-in's manifest file for each JWS server. This is inflexible and fragile if the JWS changes.

7 Related Work

Researchers have attempted to make web services more flexible as well. Most approaches relate to the dynamic composition of web services. This involves creating complex web services by dynamically integrating several simple web services together. In Sam et al.'s (Sam et al. 2006) implementation of this approach web service clients provide a specification of the web service they require to a registry service. If there are no web service providers that match the specification exactly, the registry returns the closest match. Then additional web services are used to fill the gaps in functionality and make the original web service match the user's specification more closely.

Sam et al. use the example of a Japanese tourist wanting to book a hotel in France via a web service. The registry returns a web service that allows the tourist to book French hotels however, it is in French and uses Euros as its currency, which is meaningless for the Japanese tourist. Intermediary web services are used to convert French to Japanese and Euros to Japanese Yen.

Although the hotel booking web service is made more flexible, it relies on these intermediary web services to already exist in the registry. Even if an intermediary web service does exist it may not fit the user's requirements exactly. In contrast, the JWS allows users to modify the JWS's functionality by uploading a plug-in. Since users can develop their functionality their requirements are matched exactly.

Another project looking at dynamic web services is the "Web Services Management Layer" (WSML) project (Verheecke et al. 2003). This project aims to remedy the problems caused by the traditional "Wrapper" approach to web services design, which is a static methodology, used in development environments such as Microsoft .NET. The WSML project creates a layer in between the web application and web service. This provides a layer of abstraction that can be used to allow "hot-swapping" of web services at run time. Different modules are present in this

layer, controlling areas such as security and transaction management (Verheecke et al. 2003).

The idea of making the execution of program code a web service comes from a project on verifying dynamic reconfigurations of systems (Warren et al. 2006). This project extended OpenRec, a framework that comprises a reflective component model plus an open and extensible reconfiguration management infrastructure, by integrating it with the Alloy analyser (Jackson 2002). The integration of the existing OpenRec framework, written in Python, and Alloy, written in Java, was achieved by delivering Alloy as a web service.

8 Conclusions

The key objectives of this work were to create a service that is interoperable and flexible enough to allow new functionality to be added without down time and secure enough to prevent harmful behaviour from occurring at the server. JWS meets these goals through it being a web service and through its plug-in architecture and ability to compile and execute submitted Java code. Plug-ins can be submitted and will become available at run-time without affecting the server. A further advantage of its plug-in architecture is that plug-ins are available to any client of JWS, giving further opportunities for reuse.

Plug-ins are invoked through a simple scripting language. Scripts allow users to specify which plug-ins are invoked and the order of execution of these plug-ins. Simple conditional processing is provided. Whether such a simple language is sufficient for all envisioned uses of JWS is the subject of future work.

Security of the JWS is implemented through use of the Java Security Framework. The minimum number of permissions that still allow the JWS to function are given. No permissions are currently given to uploaded plug-ins, so no uploaded code can damage the server. The only security concern in the current implementation is that denial-of-service is possible through the submission of a long-running plug-in, something that could be addressed through alternative implementations. Indeed, the security policies of the current implementation probably reduce the flexibility of JWS more than is necessary, and providing a more flexible security model is also the subject of future work.

We are not the first to consider how to provide more flexibility in web services, but we have taken a different approach to other projects, namely providing a mechanism to give the user more control over the functionality provided by the web service. We believe our minimalistic design provides a convincing proof-of-concept that our approach is viable.

References

Apache (2006*a*), 'Apache Tomcat', Apache Software Foundation `http://tomcat.apache.org`.

Apache (2006*b*), 'Web services - Axis', Apache Software Foundation `http://ws.apache.org/axis`.

Booth, D. et al. (2004), 'Web services architecture: W3c working group note 11 february 2004', Retrieved 23rd August, 2006 from `http://www.w3.org/TR/ws-arch`.

Flowers, T., Carver, C. & Jackson, J. (2004), Empowering students and building confidence in novice programmers through gauntlet, *in* 'Proceedings of the 34th ASEE/IEEE Frontiers in Education Conference', pp. 10–13.

Gamma, E. & Beck, K. (2004), *Contributing to Eclipse: Principles, Patterns, and Plug-Ins*, Addison-Wesley.

Globus (2006), 'Globus', The Globus Alliance `http://www.globus.org`.

Gong, L. (2002), 'Javatm 2 platform security architecture', Retrieved April 26, 2006 from `http://java.sun.com/j2se/1.5.0/docs/guide/security/spec/security-spec.doc.html`.

Jackson, D. (2002), 'Alloy: a lightweight object modelling notation', *ACM Trans. Softw. Eng. Methodol.* **11**(2), 256–290.

Lang, B. (2002), Teaching new programmers: A Java toolset as a student teaching aid, *in* 'Proceedings of the Inaugural Conference on the Principles and Practice of Programming', pp. 95–100.

Open, O. (2006), 'Oasis web services security (wss) tc', Retrieved August 27, 2006 from `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss`.

Sam, Y., Boucelma, O. & Hacid, M.-S. (2006), Web services customization: a composition-based approach, *in* 'ICWE '06: Proceedings of the 6th international conference on Web engineering', ACM Press, New York, NY, USA, pp. 25–31.

Verheecke, B., Cibrán, M. A. & Jonckers, V. (2003), AOP for Dynamic Configuration and Management of Web Services, *in* 'The International Conference on Web Services - Europe', pp. 137–151.

Warren, I., Sun, J., Krishnamohan, S. & Weerasinghe, T. (2006), An automated formal approach to managing dynamic reconfiguration, *in* 'ASE '06: Proceedings of the 21st IEEE International Conference on Automated Software Engineering (ASE'06)', IEEE Computer Society, Washington, DC, USA, pp. 37–46.

An Investigation on a Community's Web Search Variability

Mingfang Wu **Andrew Turpin** **Justin Zobel**

School of Computer Science and Information Technology
RMIT University
Melbourne, Australia
Email: {mingfang.wu, andrew.turpin}@rmit.edu.au, jz@acm.org

Abstract

Users' past search behaviour provides a rich context that an information retrieval system can use to tailor its search results to suit an individual's or a community's information needs. In this paper, we present an investigation of the variability in search behaviours for the same queries in a close-knit community. By examining web proxy cache logs over a period of nine months, we extracted a set of 135 queries that had been issued by at least ten users. Our analysis indicates that, overall, users clicked on highly ranked and relevant pages, but they tend to click on different sets of pages. Examination of the query reformulation history revealed that users often have different search intents behind the same query. We identify three major causes for the community's interaction behaviour differences: the variance of task, the different intents expressed with the query, and the snippet and characteristics of retrieved documents. Based on our observations, we identify opportunities to improve the design of different search and delivery tools to better support community and individual search experience.

Keywords: Web Search, Search Context, Search Log Analysis, Community Search Behaviour.

1 Introduction

A major limitation of traditional information retrieval systems is that they focus on queries and documents, and neglect the users of the systems. This is primarily because the relationships between queries and documents, and the relationships among documents, are much easier to capture, model, and compute than relationships among queries, documents, and a user's search context. Consequently, documents are retrieved because of evidence such as that they contain the query words, and are frequently referred to by other authors in the web context, instead of matching users' search intentions. This often leads to unsatisfactory search experiences.

Recently, there is a trend to improve traditional information retrieval by leveraging users' actions. This includes explicitly asking users to give relevance scales to retrieved documents (White et al. 2001), and implicitly capturing users' interactions with retrieval systems, such as eye tracking (Granka et al. 2004, Joachims et al. 2007) and desktop application monitoring (Budzik & Hammond 2000)). Among these methods, one of the most popular is trying to predicate a user's search intention/interest through mining past *clickthrough data* from Web based search engines. Clickthrough data includes past queries issued by users, the set of retrieved pages for those queries, and the set of pages chosen (clicked) for viewing by users from the list of search results.

The premise behind exploiting clickthrough data is that past interaction history could reveal a user's current search intention, assuming that there was a good reason for the user to click on a link and visit a page. If, after reading the snippet of that page, the user clicks on a document because they find the page worth further investigation, then the "click" action could be interpreted as an implicit relevance judgment of the page. As such, this clickthrough data could help to reformulate the user's current query, or re-rank current search results. By incorporating this clickthrough data into retrieval and ranking criteria, the original query and document based ranking can be enhanced with users' contextual information. Of course, the click could be accidental, or the user was distracted from the original information need and clicked a page that was off topic, and so on, so clickthrough data must be employed with caution.

When we collect and use users' clickthrough data, we can treat each user as an individual, and personalise a user's current search result based on the user's own action; alternatively, we can aggregate the clickthrough data from the communities with which the user is assoicated, and use this information to tailor search results to meet the information needs unique to that user's community. Community clickthrough data has been used in collaborative filtering (Wang et al. 2006) and social recommendation (Smyth et al. 2004). The underlying assumption is that users from a community have similar backgrounds and like minds, thus their needs for information behind a query are likely to be similar, hence a document clicked by many users in the past should be useful to future searches of the same query by other users from that community.

However, this assumption has yet to be validated: would users from a community who submitted the same queries have the same information needs? If so, would they be interested in the same set of search results? In this paper, we present a study that examines this assumption.

Through studying a community's search history in its everyday search environment, we aim to investigate the variability of a community's search behaviour and identify key factors that could influence search performance within a community. We selected a well-defined community whose members are students (or staff) from the School of Computer Science of our University. The clickthrough data (including the queries and their associated clicks) were captured through a web proxy over a period of more than 9 months. The data log captured 540,424 queries (327,064 are unique) to Google search engine. To reflect the shared interests of the community, we only extracted a set of 135 queries where each query has been issued by at least ten users. We have also restricted the queries to those occurring in the computing domain; in our investigation, queries of a personal or non-computing nature were ignored. We believe that the findings from this real community with a large set of clickthrough data would help us to reveal and understand better the nature of community search, thus allowing better informed design of information search and delivery tools for community-based search.

We present a review of the background and related work in Section 2 followed with a description of the cache log used in this study. Section 4 explores a series of research questions; and Section 5 discusses our findings and their implications to the design of information search and delivery tools. Finally Section 6 concludes the paper.

2 Background and Related Work

Substantial research, in particular from the information science perspective, has investigated users' information needs, search behaviours and processes, and perceptions of relevance (Dervin & Nilan 1986, Ingwersen 1992, Saracevic 1997), with the aim of better understanding how humans process and retrieve information. The findings from these research areas played an important role in the design and development of traditional information retrieval systems and current web search engines. Of the most relevance to our work are studies on web search context: how users search the web, what they are searching for, what the characteristics of their search queries are (Jansen et al. 2000, Spink et al. 2001), how users with different search expertise, domain knowledge, and cognitive approach search the web (Hoelscher 1998, Kim & Allen 2002), and which features of web pages may influence users' search tasks (Tombros et al. 2005). Broder (2002) and Rose & Levinson (2004) analysed users' goals for searching the web and developed a web search taxonomy to classify such goals. These studies provide a broad understanding of how the general population use web search tools, and the requirements for a search engine to satisfy this web population.

At a lower level of investigation, clickthrough data is used to implicitly capture an individual's search context, with a view to using this information to personalize search results. This method assumes that a user's past queries and their associated clicks would reveal the user's interests, and thus it could be used to predict the user's future preference. This clickthrough data can be used to model a user's immediate information need or long term preferences, depending on the period of time over which the clickthrough data was captured. For example, Shen et al. (2005) infer a user's immediate information need by her recent queries and the snippets of clicked search results. When this model of a user's short term interests is updated by a new query or click on a new page, the user's longer-term interests could be inferred (Sugiyama et al. 2004).

When the clickthrough data from users with similar information needs is aggregated, it could plausibly be used to tailor search results for the members of that community. By doing so, the privacy of individual users is also protected. Here the community refers to groups who share similar interests or information needs. This community may be predefined; for example, because the members of the community have the same or similar social background, such as a same job role in a working environment; or could be interest-based, for example dynamically inferred through users' search history (Almeida & Almeida 2004).

The usual way to use a community's clickthrough data is to treat a click associated with a query as a vote for the page's relevance. For example, Smyth et al. (2004) used a hit matrix that records the relative click frequency of retrieved pages per query, and this information is used to re-rank future search results for the same query or similar queries. They found that the current users using lists reordered with their approach could answer more fact finding questions in a given time limit, and that more questions are answered correctly.

In some work the boundary between the personal search history and community search history is blurred. In the methods of Agichtein et al. (2006) and Joachims (2002), ranking algorithms are trained based on aggregated search history obtained over a large number of users. The search history includes not only the usual clickthrough data, but also fine-grained features such as query-text features, and browsing features such as page dwell time.

Almost all of these studies (and many others that can not be mentioned here due to space considerations) report positive results, and the use of clickthrough data is a key component of all of these methods. It is natural to ask whether clickthrough data is sufficient reliable as an indicator of user preference. Would a user's own past search history or a community's search history predict the user's current interests? Teevan et al. (2005) show that a group of people from the same company and with similar IT background had different intents even when they issued the same query to a search engine, and thus they rated the retrieved pages differently. Joachims et al. (2007) conducted a controlled experiment to study the reliability of clickthrough data through manipulating the relevance ordering of search results and comparing explicit feedback against manual relevance judgment. They concluded that clicks are informative, but biased.

However, there are few studies on the characteristics of a community's search behaviour. In this study, we examine such search behaviour by analysing click-through data from a well-defined community. We focus on users' search variability behind the same queries, and the factors that may cause search variations.

3 A Community Web Cache

The data set used for our study is originally from the cache logs from our school's web proxy server. This log recorded all web activities of those students and staff for the period from 1st January 2006 through to 6th October 2006.

User Identification One of the difficulties in search log analysis is that de-identification of data to protect privacy can remove information from the log. Most studies (Spink et al. 2001) use IP addresses as an identity marker, but, in a shared computing area, a computer can be used by many different users, and a user can have access to many computers. In our case, according to the school's policy, users need to use their personal identifier to access the web. This enables us to assign each activity clearly to a distinguishable individual, and thus we can trace an individual's search history. To preserve privacy, each user's account information was replaced with an anonymous ID prior to us receiving the data.

Search Session Identification We divided the data set into search sessions. In principle, a search session should start when a searcher submits a query and end when the searcher gets information to satisfy her need or otherwise gives up the search. However, it can be difficult to rigorously detect such search session boundaries automatically from query logs. Previous studies have used various timeout periods for session segmentation, ranging from 15 minutes (He et al. 2002) or 30 minutes (Mat-Hassan & Levene 2005) to a whole day (Spink et al. 2001) according to different research goals. As our purpose of using a session is to identify those search activities of a query, we examined our data and found 15 minutes to be a reasonable boundary — our users usually shifted their search topics within 15 minutes. Later, for our targeted queries, we combined neighbouring 15 minute sessions manually where we believed a search session may have been split by the 15 minute cut-offs.

Query Statistics The Google search engine is the most frequently used search engine in our proxy logs, hence we focus our attention to those queries sent to Google. There are 540,424 Google queries in the collection (after removing empty queries and those queries from subsequent result page requests). These queries were submitted by 3,574 unique users. On average, each user submitted 151 queries over the 9 month period. The average query length is 2.64 words. This is very close to that of the general web user population (Spink et al. 2001).

Among the 540,424 queries, 260,786 (48.3%) were issued only once, with the remainder (51.7%) re-occurring at least once. Overall, there are 66,279 distinct re-occurring queries, so on average each of these was submitted 4.2 times. Table 1 shows the number of users that submitted each re-occurring query. We can see that 70% of queries that occurred more than once in the log were always submitted by the same person (though a different person for each recurring query; for example, one user issued the query "bbc news" once or twice every day), while the other 30%, which occurred multiple times in the log, were issued by more than one user.

Data Set The users recorded in our cache log searched a wide range of topics from sports, music, news, computers, science and so on. Although the majority of them have the same study major and in a similar age group and economic status, their interests as expressed in searched topics were diverse. However, they are expected to form a close-knit community when they search on the topics related to their studied major, that is, computer science and information technology. For example, when searching a particular topic, it is likely that they were taking the same lecture or doing the same assignments.

We selected queries that can meet the following criteria: 1) the query is in the computer science and information technology domain and was submitted to the Google search engine; 2) the query has been sent by at least ten users; and 3) the search sessions can be reconstructed.[1] In the end, we collected 135 such queries.

To identify the clicks associated with this set of queries, we first located all search sessions that had any of these queries, then cleaned up the following clicks by filtering out those pages that either resulted from browsing actions within a site, or were not associated with the selected query. After the clean-up, we found that these 135 queries were searched 3,480 times by 1,115 users, each query re-occurring 25.8 times on average. There were 4697 clicks in total — 1.3 clicks per search on average. There were 14.9%, 56.5%, and 28.6% searches that had zero clicks, one click, and more than one click respectively. The reasons behind the queries with zero clicks are unknown; the searchers could be satisfied (or unsatisfied) by looking at snippets only, or users were handling multiple tasks (Spink et al. 2006) at the same time.

4 Community Search Analysis

4.1 Overall Click Behaviour

It is often assumed that a user's action of clicking on a URL indicates the page's relevance to the submitted query. However, studies of user clicking behaviour show that, while a user's decision to click is mainly influenced by the relevance of the snippets associated with the pages, it may also be biased due by the order

[1] As the retrieved document set was not recorded at the time when a user issued the query, we reconstructed this set by sending the same query to the Google search engine at a later date (from 2nd December 2006 to 7th December 2006), and recorded the URLs of the top ten retrieved pages. In order to minimise the differences between the original and reconstructed retrieved sets, we don't include queries where most of their clicked pages are not in the reconstructed top ten list. Here we take ten as a threshold because previous studies showed that most users do not access search results past the first page (Jansen et al. 2000).

No. of users	1	2	3	4	5	6	7	8	9	10+
Queries (%)	70.2	19.9	4.3	1.8	1.0	0.6	0.4	0.3	0.2	1.1

Table 1: The proportion (%) of users responsible for re-occurring queries.

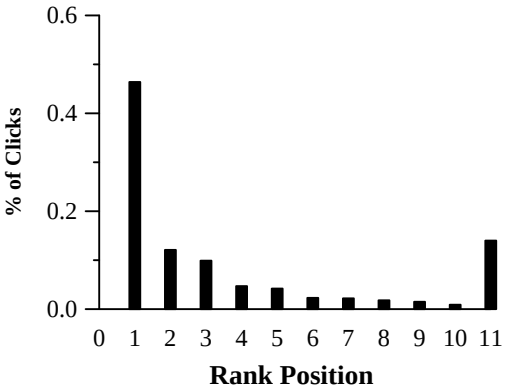


Figure 1: Percentage of clicks at each rank position.

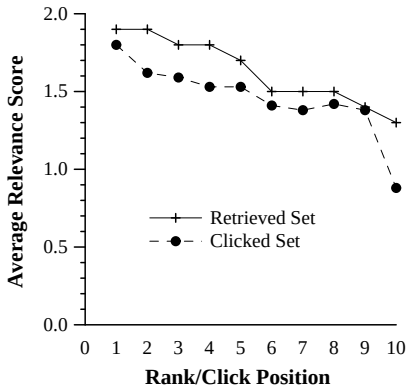


Figure 2: Average relevance score at each position in two ranked lists.

of a page in a ranked list of search results (Joachims et al. 2007). In this section, we presented our investigation if the above claims still hold true for our selected close-kit community. In particular, we set out to investigate the following three research questions.

1. Is it true that a page with higher rank would be clicked by users more frequently than a page with lower rank?
2. Does a ranking of documents based on *click frequency* correlate with a *relevance*-based ranking?
3. Are clicked pages relevant?

Q1. Is it true that a higher ranked page would be clicked by users more frequently?

For each page in the search result set (10 per query for 1350 in total), Overall, there are 762 retrieved pages that have zero clicks, and their average rank is 6.5, while the average rank of the other 588 pages with at least one click is 4.1. The ranks of clicked pages is significantly higher than that of un-clicked pages (un-paired t-test, $p < 0.01$). This shows that those clicked pages have higher rank on average.

Figure 1 shows the distribution of clicks for each ranked position. Clicks that do not select a page in the top ten answers for a query are assigned rank eleven. Nearly half of the clicks (46.7%) are on the top-ranked URL, 12.3% of clicks are on the second-ranked URL, and 14% of clicks are at 11th position. These figures indicate that higher-ranked pages are selected more frequently than lower-ranked pages, confirming that our query log shares characteristics of that used by Joachims et al. (2007).

Q2. Does ranking based on click frequency correlate with relevance-based ranking?

We re-ranked each of the top ten search results in the order of their click frequency from high to low, and use Kendall's-τ rank correlation coefficient to measure the degree of correspondence between two ordered lists. The Kendall's-τ coefficient ranges from -1 (perfect disagreement — one list is the reverse of the other), through 0 (the ranks of two lists are independent), to 1 (perfect agreement — the ranks of the two lists are exactly the same). As there are ties in the re-ranked list, the Kendall τ_b is used (Fagin et al. 2003).

Among the 135 queries, there are 86 queries whose paired lists have strong tendency of agreement ($\tau_b >= 0.4$, of which 40 are significant at $p < 0.05$ level, by the two-tailed Z-test), and for the other 49 queries the paired lists are independent ($|\tau_b| < 0.4$). There was no perfect disagreement found for any query.

Q3. Are the clicked pages relevant?

The assumption behind utilizing clickthrough data for ranking is that a click is a form of relevance judgment — the clicked pages are more relevant than those not clicked. Did our users click on a page selectively or just click on the top-ranked pages? To answer this question, we examined the relevance of the top ten pages from each search result.

Each page was judged for relevance (by either one of authors or a postgraduate student) on a three-point scale: highly relevant, relevant and irrelevant. These corresponded to scores of 2, 1, and 0 respectively. Figure 2 shows the mean relevance scores over all search sessions for the set ranked using click frequency, and the set ranked as per the Google result. We can see that the clicked data are of high relevance to the queries. Over 49,240 clicks, there are only 591 (12%) that were judged to be on an irrelevant page.

We have seen that our users tend to click the top-ranked search pages, and Figure 2 shows that the top ranked pages are also of high quality. A remaining question is whether our users clicked the top-ranked search results blindly or clicked relevant URLs that happened to be highly ranked.

We have only three queries in our data set whose first ranked page was judged irrelevant. We found that, in these cases, the majority of our users clicked lower ranked, yet relevant, pages. For example, for the query "ssi", the relevant pages appear at positions 4, 5, and 7 (the pages at position 4 and 5 are from the same site). Of the 25 users that issued this query:

- 11 users did not click any page;
- 6 users clicked on the fourth or seventh ranked page;
- 2 users first clicked on the first ranked page and then clicked on the fourth search page; and
- For the remaining 6 users, their first click is not on the top ten list but 5 of them are judged relevant or highly relevant.

For the 11 sessions without any click, there are query modifications in 10 sessions to either expand the query to "server side includes" or include more contextual words such as "ssi in html". In these cases, we assume that the users make relevance judgments by reading the snippets only. From these observations and the finding from a systematic evaluation (Thomas & Hawking 2006) that users were able to reliably distinguish between high- and low-quality result sets, we can be confident that our users did not click a relevant page just by chance.

4.2 User Click Variability

From the above discussion, we see that users demonstrated a tendency to click on highly ranked documents. However, we observed a difference in click patterns among users even for the same query. Here we measure this difference by using inter-rater agreement[2]. We calculate it in two ways. First, we treat the clicks from a query as a simple *click-list* ignoring the position of the click in the ranked list. The average inter-rater agreements over all queries is 0.36. That is, only about a third of all possible pairs of users over all queries clicked the same set of pages.

Second, we calculate the inter-rater agreement amongst the first click made by all users, then the second click, and so on (we refer to this as the click position of the click, as opposed to the click rank). As shown in Figure 3, the inter-rater agreement for click position 1 is 0.52. It then dropped dramatically to 0.25 for click position 2. The decreasing inter-rater agreement as click position increases indicates that users have a tendency to click the top-ranked page (hence the high agreement for click position one) and then accessed the remaining search results in different orders. This could be because they interpret the snippets on the results pages differently, or because their search intentions differ for identical query strings.

4.3 Task Variability

To understand why the members of this close-knit community sent the same query but clicked on different pages, we scrutinised those queries and their associated clicks. We found that search task variation

[2]Inter-rater agreement gives a score of how much consensus there in the ratings given by judges. Here we treat each user as a rater, and her click on a URL as a judge.

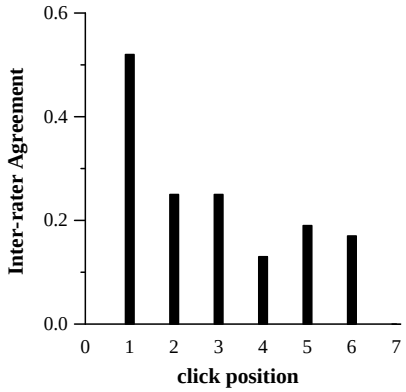


Figure 3: Inter-rater agreement at each click position.

Query type	Examples
navigational	cygwin
	delicious
	java api 1.5
	apache
	ctrl alt del
broad informational	big o notation
	css tutorial
	software requirement specification
	project management
	bioinformatics
	xml schema
specific informational	c printf
	matrix multiplication
	ascii chart
	sql max
	php date

Table 2: Examples of queries in each categories

is one of the factors that may cause the click difference. Referring to search task classification in the web search context (Broder 2002, Rose & Levinson 2004), we classified our queries into three categories: navigational queries, specific informational queries, and broad informational queries. Here the navigational queries are those whose intent is to reach a particular site (for example, "cygwin" and "delicious", while the informational queries are aimed to acquire information or facts that may be contained in one or more web pages (which are unknown to the user). Here we further divide the informational queries into two groups: specific and broad information queries, depending on whether a query implies a multitude of facets or interpretations. For example, "binary tree", "c tutorial", and "test plan" would be regarded as broad informational queries while "binary search tree in c", "c strtok", and "sql max" are specific informational queries. Table 2 shows some example queries in each of categories.

We conjecture that users' click patterns may be more similar for navigational and specific informational queries, and less on broad informational queries, on the grounds that the former two types of queries embody a clear information goal, while the motivations behind the broad informational queries may be quite diverse and rich. A user might want to investigate a broad topic ("c tutorial"), or a user

	Inter-rater agreement			
	High	Medium	Low	Overall
Navigational	12	2	0	14 (14.4%)
Specific Inf.	17	12	3	32 (23.7%)
Broad Inf.	9	30	50	89 (66.9%)

Table 3: Number of queries categorised by type and inter-rater agreement of click-patterns. High inter-rater agreement is larger than 0.7, Medium is between 0.4 and 0.7, and Low is less than 0.4.

could target a particular concept ("c strtok") as part of a typical search strategy that ranges from broad to narrow (Spink et al. 2001). Alternatively, a user may first issue a specific informational query, but, on receiving poor results, then re-issue a broad query.

Table 3 shows the distribution of queries at each level of inter-rater agreement (of click-lists) among the three query categories. Overall, there are about 10%, 23%, and 67% of the queries in navigational, specific informational, and broad informational categories respectively. In the high inter-rater interval, where agreement is at least 0.7, 76.3% of queries are in the navigational and specific informational query categories, as opposed to the low inter-rater interval (agreement less than 0.4), where 94.3% queries are broad informational queries.

Note that some broad queries have unexpectedly high inter-rater agreement. We examined the search results and clicked pages for these nine queries and found that all these queries lead to clicks on comprehensive resource pages (four of them are `www.w3.org`); these pages provide easy-to-navigate links to most facets of a topic domain, and so users with diverse information needs can find their information through navigation rather than search. This also indicate that a user's click behaviour is also influenced by the characteristics of a retrieved page.

Three specific informational queries have low inter-rater agreement, and it is not obvious why this is the case. We suspect that the difference in snippet quality for these queries may be the major reason. For example, for the query "c strtok" the users' clicks are divided into the top three documents, which all have the same quality of information (example and explanation), yet the snippets of the three documents are slightly different as shown in Figure 4. The first is highly generic, while the second shows an example code line and the third has a problem diagnosis. This may explain why some users skipped the first and clicked either the second or the third.

We find that both navigational and specific queries have significantly more clicks than broad information queries (un-paired two tailed t-test, $p < 0.03, 0.002$, respectively); the low inter-rater category also has more clicks than the high-inter category, as shown in Table 4. Figure 5 also shows that the click distributions for each query type are different: the clicks from navigational queries and specific information queries are skewed towards the top-ranked page and the top three pages respectively, while the clicks from broad information queries are scattered, although the top-ranked pages attract more clicks.

Tables 5 and 6 also show that the average ranks of first clicks and all clicks for the broad informational

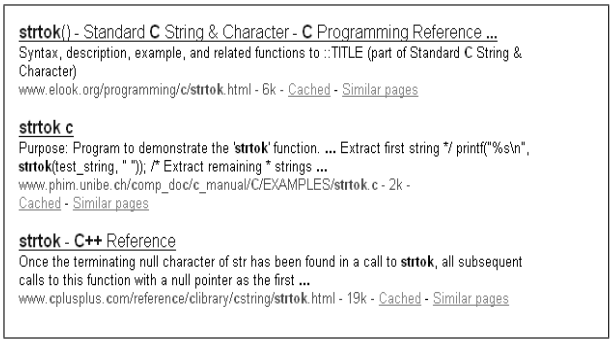


Figure 4: The snippets for the top three results for the query "c strtok".

Inter-rater	High	Middle	Low	Overall
Navigational	1.44	1.36		1.43
Specific Inf.	1.32	1.57	1.47	1.43
Broad Inf.	1.61	1.62	1.79	1.72

Table 4: Average number of clicks per query.

queries are significantly lower than the other two categories ($p < 0.0001$), even more so for the broad informational queries with low inter-rater agreement. This may indicate that these two measures could be taken to identify query types, thus allowing application of different search and relevance feedback strategies to the queries of each category.

4.4 Query Reformulation Variability

Given that our users show different search patterns for different tasks but can still find a set of relevant retrieved pages, does the set of relevant pages satisfy our user's information needs? Relevance of a page can range from topic relevance and situational relevance to cognitive relevance (Saracevic 1996). The relevance judgements we have used in this study are at the topic relevance level, that is, whether a search result is relevant to the search query topic — a TREC-like (Text REtrieval Conference) assessment (Voorhees 2005). The ultimate goal of an information system is to satisfy users' informational needs at a situational and cognitive level; that is, whether a search result is *useful* to a user's task at hand and right to her knowledge level. The best way to answer this question is to interview users at the time of search. In the absence of this information, we can estimate satisfaction by examining users' query reformulation history. We assume that if a user keeps reformulating her query, most likely the information she has so far does not satisfy her information need.

For each query in the selected query set, we located the search sessions, and manually examined whether a query has been modified. For each query, the query modification rate is calculated as the number of sessions with modified queries divided by the total number of sessions. Thus, the higher the query modification rate, the more users modified the query. Overall, the average query modification rate is 41.5%. This high query modification rate indicates that merely delivering a list of highly ranked, topically relevant

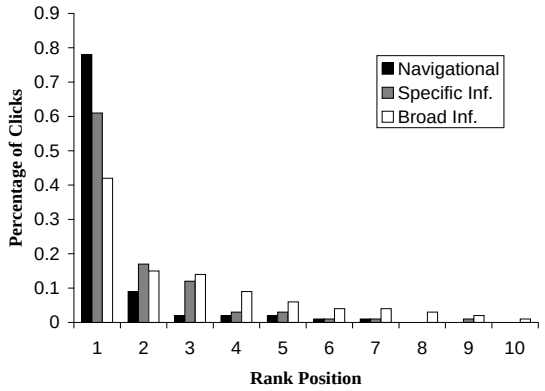


Figure 5: Click distribution at each rank position.

Inter-rater	High	Middle	Low	Overall
Navigational	1.23	1.63		1.29
Specific Inf.	1.22	2.27	2.91	1.77
Broad Inf.	1.34	2.59	4.23	3.38

Table 5: Average rank of first click.

documents is not enough to satisfy 41.5% of information needs.

By closely examining the query modification history, we found that, for queries with high query reformulation rates, the initial query is reformulated by different users into different facets of the original query topic. Take the query "test plan" as an example. This query was sent 17 times (by 12 users) and was modified 12 times. The subsequent modified queries include: "test plan template", "what is test plan", "test plan sample", "test plan technique", "test case distribution", and "test plan acceptance criteria".

A strong correlation is observed between the query modification rate and the inter-rater agreement of click-lists ($r = -0.41, t = -5.26, p < 0.01$) or agreement amongst the first clicked page ($r = -0.38, t = -4.68, p < 0.01$). Task by task, navigational and specific informational queries have significantly lower query reformulation rates than broad informational queries, as shown in Table 7. This may indicate that, if the inter-rater agreement among users' click set is low, there may be a high chance that the query carries multiple information intents.

5 Discussion and Implications

We have explored the community's overall search pattern. Our results confirm that users tend to click on top ranked pages, and consequently that those top ranked pages also have a high click frequency. A detailed analysis of users' click history revealed that the majority of users clicked on pages that are topically relevant to search queries.

However, we inferred that a users' decision to click on a page was limited by what snippets were presented. In most cases, users' clicked pages are relevant but might not be useful, as evidenced by low inter-rater agreement on clicked pages and a high query reformulation rate. We observed that different users

Inter-rater	High	Middle	Low	Overall
Navigational	1.64	1.93		1.68
Specific Inf.	1.56	2.59	3.02	2.08
Broad Inf.	1.92	3.05	4.59	3.80

Table 6: Average rank of all clicks.

Inter-rater	High	Middle	Low	total
Navigational	0.26	0.54		0.30
Specific Inf.	0.31	0.35	0.38	0.34
Broad Inf.	0.36	0.46	0.49	0.47

Table 7: Query reformulation rates in each query category

reformulated their queries into queries on different facets of their search topic. This branch out from the same query to different facets indicates that users' information needs may be different even though they have a similar background and submitted the same query.

In Section 4.3 we examined the task variation on clicking variability and identified a number of different click patterns for different search tasks. We found that users clicked significantly more pages for broad informational queries than those for navigational and specific informational queries. The rank of clicked pages from those broad information queries are significantly lower than those from navigational and specific informational queries. Users agree more on navigational and specific information queries than the broad informational queries. These findings indicate that the value of community clickthrough data varies for different search tasks.

Using clickthrough data to alter rankings will most likely benefit the specific informational search tasks and the homepage finding task, as these tasks are precision oriented and a user's information need can usually be satisfied by just one web page. If a search result list already has a high precision, then incorporating community clickthrough data may not help much, but would not do any harm either. However, if a search result list has a poor precision, then using the community's click frequency data would most likely bring relevant pages to the top of the list as the majority of community members click on the relevant page (especially when the snippet of the page is of high quality). An alternative use of the clickthrough data is to automatically expand the query by using the pages with a high click frequency.

Care should be taken when using community clickthrough data in relevance feedback for broad informational queries. For these type of queries, it may be preferable to deliver search results that cover as many facets as possible (width first). We tried a traditional relevance feedback method (using the text or snippet of clicked pages as a source for query expansion) for two broad queries. The effect was to raise the ranking of pages that are similar to the clicked pages (increasing the depth), without increasing the coverage of more facets of the searched topics. In our opinion, for this type of query, not only the relevance of a page but also the novelty of the page should be considered; and the snippets of each retrieved page should also differentiate one page from each other. Other

work (Carbonell & Boldstein 1998, Zhai & Lafferty 2003) gives examples of how to increase the diversity of search results.

To accommodate the diverse search intentions behind a broad informational query, a search system should support not only the querying activities but also the after-query browsing activity. We could use a domain specific taxonomy to categorise search results as demonstrated in DynaCat (Pratt et al. 1999). In case there isn't a ready-to-use taxonomy for a particular domain, we could use the community query reformulation history to guide the search and search result organisation. Take the query "test plan" from Section 4.4 as an example; we can derive all facets of a broad query from the community's query reformulation history. If we could take one or two top-ranked pages from each of these reformulated queries or facets to form a new list, it would implicitly show a diversified list that covers many facets. As shown in Figure 6, we may even explicitly show the pages under headings derived from the query reformulations, to give users a clearer view what has been retrieved and help users navigate this retrieved document space to get information they need. The more members of the community search on a topic, the more comprehensive an answer list would be.

Finally, to support various search tasks, a search engine should have the ability to automatically identify each query type so that it can apply the optimal ranking scheme for each task. Click distribution and anchor-link distribution have been explored to predicate a users' search goal (Lee et al. 2005). Here, in a community search context, we could use a variety of criteria to classify a query: inter-rater agreement among users' clicks, query reformulation rate, the average number of clicks, or the average rank of first click. All these measures are significantly correlated ($p < 0.01$). Different thresholds should be tested for each measure and data set. For example, in our data set, 74.9% queries with query modification rate greater than 0.4 are broad queries, and 75.0% with query modification rate less than or equal to 0.2 are navigational and specific queries; 83.3% queries with the average rank of the first click greater than 1.5 are broad queries, and 76.3% queries with the average rank of the first click less than or equal to 1.5 are navigational and specific queries.

6 Conclusion

We have explored community search history aiming to identify opportunities to better support community members' information searching tasks. We found that: users tend to click on highly ranked pages and consequently the highly ranked pages also have a high click frequency; the community shows diverse search patterns for different search tasks; and the information needs behind broad informational queries are different even for members of the close-knit community.

Our findings indicate that, the gain of using a community's search history to improve future search experience mainly from the specific informational searches and the navigational searches. For broad information searches, using clickthough data can only bring together similar pages, and this will not satisfy the diverse information needs of the community. We found that users' query reformulation history may provide a potential source for query expansion to broaden the range of web pages returned, and to organise those pages clearly to different facets to highlight the diversity and thus to support browsing activities. Further experiments with users will be necessary to determine the benefit of this claim.

In this study, we focused on the community search of web content. The characteristics of communities and searched domains may vary in other situations. In our future work, we will also investigate the search behaviour of close-kit communities with a closed set document collection, as well as the search behaviours of dynamically bonded community with various document collections.

Acknowlegements

This work was supported by the Australian Research Council.
We would like to thank Yanghong Xiang for collecting some data for this study.

References

Agichtein, E., Brill, E. & Dumais, S. (2006), Improving web search ranking by incorporating user behaviour information, *in* S. Dumais, D. Hawking & K. Jarvelin, eds, 'Proceedings of the 29st ACM-SIGIR Conference on Research and Development in Information Retrieval', Seattle, Washington, USA, pp. 19–26.

Almeida, R. & Almeida, V. (2004), A community-aware search engine, *in* M. Najork & C. Wills, eds, 'Proceeding of the 13th ACM-WWW Conference on on World Wide Web', New York, USA, pp. 413–421.

Broder, A. (2002), 'A taxonomy of web search', *ACM SIGIR Forum* **36**(2).

Budzik, J. & Hammond, K. (2000), User interactions with everyday applications as context for just-in-time information access, *in* D. Riecken, D. Benyon & H. Lieberman, eds, 'Proceedings of ACM-IUI Conference on Intelligent User Interfaces', New Orleans, Louisiana, pp. 44–51.

Carbonell, J. & Boldstein, J. (1998), The user of MMR, diversity-based reranking for reordering documents and producing summaries, *in* W. B. Croft, A. Moffat, C. J. van Rijsbergen, r. Wilkinson & J. Zobel, eds, 'Proceedings of the 21st ACM-SIGIR Conference on Research and Development in Information Retrieval', Melbourne, Australia, pp. 335–336.

Dervin, B. & Nilan, M. (1986), 'Information needs and uses', *Annual Review of Information Science and Technology* **21**, 3–33.

Fagin, R., Kuman, R. & Sivakumar, D. (2003), 'Comparing top k lists', *SIAM Journal on Discrete Mathematics* **17**(1), 134–160.

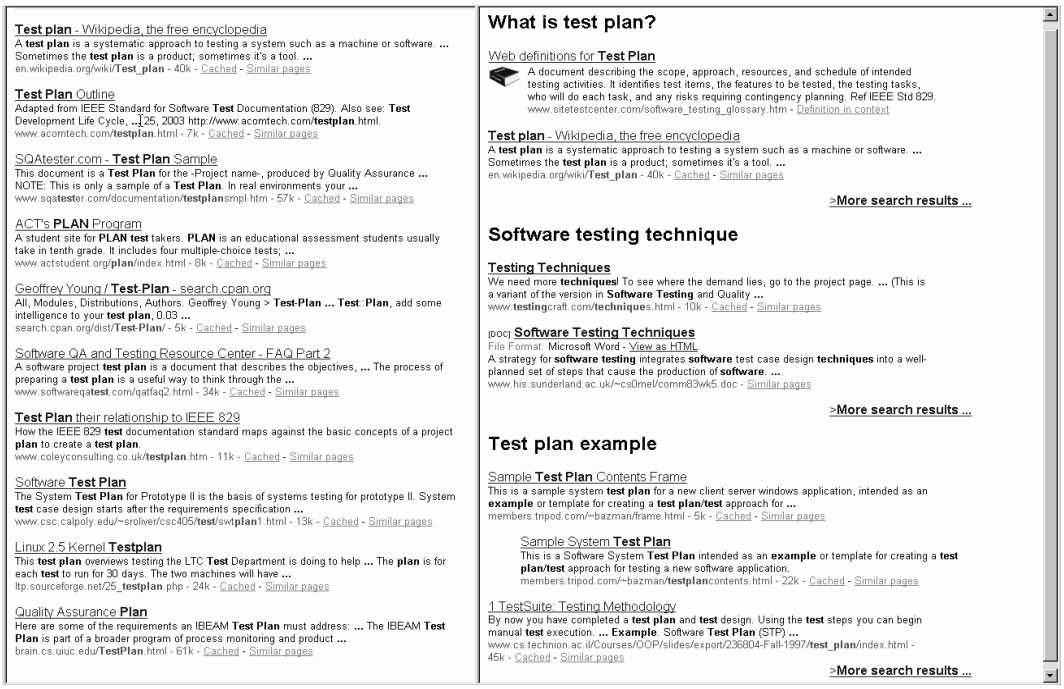


Figure 6: Using query reformulation history for comprehensive answer constructing

Granka, L. A., Joachims, T. & Gay, G. (2004), Eye-tracking analysis of user behavior in www search, *in* K. Jarvelin, J. Allan & P. Bruza, eds, 'Proceedings of the 27st ACM-SIGIR Conference on Research and Development in Information Retrieval', Sheffield, UK, pp. 44–51.

He, D., Goker, A. & Harper, D. J. (2002), 'Combining evidence for automatic web session identification', *Information Processing and Management* **38**, 727–742.

Hoelscher, C. (1998), How internet experts search for information on the web, *in* 'Proceedings of WebNet'98'.

Ingwersen, P. (1992), *Information Retrieval Interaction*, Taylor Graham.

Jansen, B. J., Spink, A. & Saracevic, T. (2000), 'Real life, real users and real needs: a study and analysis of user queries on the web', *Information Processing and Management* **36**, 207–227.

Joachims, T. (2002), Optimizing search engines using clickthrough data, *in* 'Proceedings of ACM-SIGKDD Conference on Knowledge Discovery and Datamining', Alberta, Canada, pp. 133–142.

Joachims, T., Granka, L., Pan, B., Hembrooke, H., Radlinski, P. & Gay, G. (2007), 'Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search', *ACM Transactions on Information Systems (TOIS)* **25**(2), 1–26.

Kim, K.-S. & Allen, B. (2002), 'Cognitive and task influences on web search behavior', *Journal of the American Society for Information Science and Technology* **53**(2), 109–119.

Lee, U., Liu, Z. & Cho, J. (2005), Automatic identification of user goals in web search, *in* 'Proceeding of the 14th ACM-WWW Conference on on World Wide Web', Chiba, Japan, pp. 391–400.

Mat-Hassan, M. & Levene, M. (2005), 'Associating search and navigation behavior through log analysis', *Journal of the American Society for Information Science and Technology* **56**(9), 913–934.

Pratt, W., Hearst, M. A. & Fagan, L. M. (1999), A knowledge-based approach to organizing retrieved documents, *in* 'American Association for Artificial Intelligence', pp. 80–85.

Rose, D. E. & Levinson, D. (2004), Understanding user goals in web search, *in* M. Najork & C. Wills, eds, 'Proceeding of the 13th ACM-WWW Conference on on World Wide Web', New York, USA.

Saracevic, T. (1996), Relevance reconsidered, *in* P. Ingwerson & N. O. Pors, eds, 'Proceedings of the Second International Conference on Conceptions of Library and Information Science (CoLIS): Integration in Perspective', Copenhagen: Royal School of Librarianship, pp. 201–218.

Saracevic, T. (1997), 'The stratified model of information retrieval interaction: extension and applications', *Proceedings of the American Society for Information Science* **34**, 313–327.

Shen, X., Tan, B. & Zhai, C. (2005), Context-sensitive information retrieval using implict feedback, *in* G. Marchionini, A. Moffat & J. Tait, eds, 'Proceedings of the 28st ACM-SIGIR Conference on Research and Development in Information Retrieval', Salvador, Brazil, pp. 43–50.

Smyth, B., Balfe, E., Freyne, J., Briggs, P., Coyle, M. & Boydell, O. (2004), 'Exploiting query repetition and regularity in an adaptive community-based web search engine', *User Modeling and User-Adaped Interaction* **14**(5), 383–423.

Spink, A., Park, M., Jansen, B. & Pedersen, J. (2006), 'Multitasking during web search sessions', *Information Processing and Management* **42**(1), 264–275.

Spink, A., Wolfram, D., Jansen, B. J. & Saracevic, T. (2001), 'Searching the web: the public and their queries', *Journal of the American Society for Information Science and Technology* **52**(3), 226–234.

Sugiyama, K., Hatano, K. & Yoshikawa, M. (2004), Adatptive web search based on user profile constructed without any effort from users, *in* M. Najork & C. Wills, eds, 'Proceeding of the 13th ACM-WWW Conference on on World Wide Web', New York, USA, pp. 675–684.

Teevan, J., Dumais, S. T. & Horvitz, E. (2005), Beyond the commons: Investigating the value of personalizing web search, *in* 'Proceedings PIA 2005: Workshop on New Technologies for Personalized Information Access', pp. 84–92.

Thomas, P. & Hawking, D. (2006), Evaluation by comparing result sets in context, *in* V. Tsotras, E. Fox & B. Liu, eds, 'Proceedings of the 15th ACM-CIKM Conference on Information and Knowledge Management', Virginia, USA, pp. 94–101.

Tombros, A., Ruthven, I. & Jose, J. M. (2005), 'How users access web pages for information seeking', *Journal of the American Society for Information Science and Technology* **56**(4), 327–344.

Voorhees, E. M. (2005), Overview of trec 2005, *in* 'The 14th Text REtrieval Conference (TREC 2005) Proceedings', Gaithersburg, MD, USA.

Wang, J., de Vries, A. P. & Reinders, M. J. T. (2006), A user-item relevance model for log-based collaborative filtering, *in* M. Lalmas & A. Tombros, eds, 'Proceedings of the Annual European Conference on Information Retrieval (ECIR)', London, UK, pp. 37–48.

White, R. W., Jose, J. M. & Ruthven, I. (2001), Comparing explicit and implicit feedback techniques for web retrieva: Trec-10 interactive track report, *in* 'Proceedings of the 10th Text REtrieval Conference (TREC)', Gaithersburg, Maryland, USA.

Zhai, C. & Lafferty, J. (2003), Beyond independent relevance: Methods and evaluation metrics for subtopic retrieval, *in* J. Callan, D. Hawking & A. Smeaton, eds, 'Proceedings of the 26st ACM-SIGIR Conference on Research and Development in Information Retrieval', Toronto, Cannada, pp. 10–17.

On Illegal Composition of First-Class Agent Interaction Protocols

Tim Miller[1*] **Peter McBurney**[2]

[1]Department of Computer Science and Software Engineering
The University of Melbourne, Victoria, 3010, Australia

[2]Department of Computer Science,
University of Liverpool, Liverpool, L69 7ZF, UK
p.j.mcburney@csc.liv.ac.uk

Abstract

In this paper, we examine the composition of first-class protocols for multi-agent systems. First-class protocols are protocols that exist as executable specifications that agents use at runtime to acquire the rules of the protocol. This is in contrast to the standard approach of hard-coding interaction protocols directly into agents — an approach that seems too restrictive for many intelligent and adaptive agents. In previous work, we have proposed a framework called $\mathcal{RASA}$, which regards protocols as first-class entities. $\mathcal{RASA}$ includes a formal, executable language for multi-agent protocol specification, which, in addition to specifying the order of messages using a process algebra, also allows designers to specify the rules and consequences of protocols using constraints. Rather than having hard-coded decision making mechanisms for choosing their next move, agents can inspect the protocol specification at runtime to do so. Such an approach would allow the agents to compose protocols at runtime, instead of relying on statically designed protocols. In this paper, we investigate the implications of *protocol composition* by examining the conditions under which composing existing legal protocols would lead to illegal protocols — that is, protocols that can fail during execution through no fault of the participants. We precisely define what constitutes an illegal protocol, and present proof obligations about compositions that, when discharged, demonstrate that a composition is legal.

Keywords: interaction protocols, multi-agent systems.

1 Introduction

Research into multi-agent systems aims to promote autonomy and intelligence into software agents. Intelligent agents should be able to interact socially with other agents, and adapt their behaviour to changing conditions. Despite this, research into interaction in multi-agent systems is focused mainly on the documentation of interaction protocols, which specify the set of possible interactions for a protocol in which agents engage. Agent developers use these specifications to hard-code the interactions of agents. We identify three significant disadvantages with this approach: 1) it strongly couples agents with the protocols they use — something which is unanimously discouraged in software engineering — therefore requiring agent code to changed with every change in a protocol; 2) agents can only interact using protocols that are known at design time, a restriction that seems too restrictive for many intelligent and adaptive agents; and 3) agents cannot compose protocols at runtime to bring about more complex interactions, therefore restricting them to protocols that have been specified by human designers — again, this seems too restrictive for many intelligent and adaptive agents.

In previous work [9, 10], we have proposed a framework called $\mathcal{RASA}$, which regards protocols as *first-class* entities. These first-class protocols are documents that exist within a multi-agent system, in contrast to hard-coded protocols, which exist merely as abstractions that emerge from the messages sent by the participants. To promote decoupling of agents from the protocols they use, we propose a formal, executable language for protocol specification. This language consists of a process algebra, used to specify the sequencing of messages. The messages are represented as atomic actions, and each atomic action contains the *rules* governing under which conditions the message can be sent, and the *effects* that sending the message has on a system. Rather than a protocol specification being just a sequence of arbitrary tokens, these rules and effects give the protocol *meaning*, and the rules and effects of the overall protocol can be derived compositionally from the meaning of the atomic protocols that comprise it. Instead of hard-coding the decision process of when to send messages, agent designers can implement goal-directed agents that reason about the effect of the messages they send and receive, and can choose the course of action that best achieves their goals, allowing agents to learn new protocols at runtime, and maintain libraries of protocols through which they can search to find the protocols that best achieve their goals. In addition, agents would be able to compose new protocols from existing protocols at runtime if they know of protocols that achieve their goals.

Composing protocols, whether statically at design time, or dynamically at runtime, is not an arbitrary task. Clearly, the composer must consider whether the composite protocol will go some way to achieving its goals. Determining this is a domain-specific (and most likely, agent-specific) problem, in which different agents will have different ideas within different environments or at different times as to whether a composite protocol is suitable.

However, there are certain conditions in which a protocol composition results in a protocol that can lead to a runtime error, which we categorise as illegal protocols. In this paper, we define what it means for a protocol to be illegal, and then identify the properties that must hold for protocols to be legal. We provide proof obligations for each type of composi-

[*]The work published in this paper was carried out while Tim Miller was at the Department of Computer Science, University of Liverpool.

tion that, when discharged, imply that a protocol is legal, and then identify how to verify a composition from existing legal protocols using these proof obligations. These proof obligations should be discharged every time a composition is performed, whether statically or dynamically. Emphasis is placed on protocols specified in the $\mathcal{RASA}$ protocol language, but such ideas would be applicable to protocols specified in any language with features similar to $\mathcal{RASA}$'s.

The outline of this paper is as follows. Section 2 presents a brief overview of the $\mathcal{RASA}$ framework, and an example protocol specified using the $\mathcal{RASA}$ language. Section 3 formally defines our notion of an illegal protocol composition, and Section 4 presents the proofs the must be discharged to verify that a composition is illegal, and a method for discharging these. Section 5 presents related work and Section 6 concludes the paper.

2 $\mathcal{RASA}$ Overview

In this section, we present a brief overview of the $\mathcal{RASA}$ framework.

2.1 Modelling Information

Communication in multi-agent systems is performed across a *universe of discourse*. Agents send messages expressing particular properties about the universe. We assume that these messages refer to *variables*, which represent the parts of the universe that have changing values, and use other *tokens* to represent relations, functions, and constants to specify the properties of these variables and how they relate to each other.

Rather than devise a new language for expressing information, or using an existing language, we take the approach that any constraint language can be used to model the universe of discourse, provided that it has a few basic constants, operators and properties.

We use the definition of a *cylindric constraint system* proposed by De Boer *et al.* [2]. They define a cylindric constraint system as a complete algebraic lattice, $\langle C, \sqsupseteq, \wedge, \text{true}, \text{false}, Var, \exists\rangle$. In this structure, C is the set of atomic propositions in the language, for example $X \leq Y$, $\sqsupseteq$ is an entailment operator, true and false are the least and greatest elements of C respectively, $\wedge$ is the least upper bound operator, Var is a countable set of variables, and $\exists$ is an operator for hiding variables. The entailment operator defines a partial order over the elements in the lattice, such that $c \sqsupseteq d$ means that the information in d can be derived from c. The shorthand $c \equiv d$ is equivalent to $c \sqsupseteq d$ and $d \sqsupseteq c$. We will use $\mathcal{L}$ to refer to the language, as well as the set of all constraints in the language; for example, $c \in \mathcal{L}$.

A constraint is one of the following: an atomic proposition, c, for example, $X = 1$, where X is a variable; a conjunction, $\phi \wedge \psi$, where ϕ and ψ are constraints; or $\exists_x \phi$, where ϕ is a constraint and $x \in Var$. We extend this notation by allowing negation on the right of an entailment operator, for example, $c \sqsupseteq \neg d$, which is equivalent to $c \not\sqsupseteq d$. Other propositional operators are then defined from these: $\phi \vee \psi \mathrel{\widehat{=}} \neg(\neg\phi \wedge \neg\psi)$, $\phi \rightarrow \psi \mathrel{\widehat{=}} \neg\phi \vee \psi$, and $\phi \leftrightarrow \psi \mathrel{\widehat{=}} \phi \rightarrow \psi \wedge \psi \rightarrow \phi$. We will continue to use the symbols ϕ and ψ to refer to constraints throughout this paper.

We introduce a renaming operator, which we will write as $[x/y]$, such that $\phi[x/y]$ means 'replace all references of y in ϕ with x'. The reader may have already noted that $\phi[x/y]$ is shorthand for $\exists_y(y = x \wedge \phi)$.

2.2 Modelling Protocols

The $\mathcal{RASA}$ protocol specification language is based on process algebras, and resembles languages such as CSP [7]. However, we add the notion of *state* to the language. State is useful, because it allows us to build up the meaning of protocols compositionally, for example, the effect of sending two messages is the effect of sending the second in the state that results after sending the first. The final outcome of the protocol is the end state. A detailed presentation of the specification language, including operational semantics, is available in [10].

A protocol specification is a collection of protocol definitions of the format $N(x, \ldots, y) \mathrel{\widehat{=}} \pi$, in which $N, x, \ldots, y \in Var$, and π represents a protocol.

Protocols are defined using the two types of atomic protocol, and algebraic operators for building up compound protocols from these. The first atomic protocol is message sending, contains three parts, and is specified like so: $\psi \xrightarrow{c(i,j).\phi_m} \psi'$. This is read as follows: if the precondition, ψ, is provable from the current state, then the agent i is permitted to send the message ϕ_m, or any message ϕ'_m, such that $\phi'_m \sqsupseteq \phi_m$, to agent j. The effect of this message on the state is specified by the postcondition, ψ'. We allow agents to send the message ϕ'_m, such that $\phi'_m \sqsupseteq \phi_m$, so that agents can further constrain the values of the messages; thus, ϕ_m is only a template of the message. Omitting the prefix $c(i, j)$ from a message template implies that ϕ_m is an action that must be performed. In this paper, we omit (i, j) when we do not care who the sender and receiver of the message is. We use the notion of *inertia* in calculating the new state from the postcondition; that is, any variables in ψ' are constrained by ψ' in the new state, and any other variables in the state are left unchanged. The second atomic action/protocol is the empty action, the syntax of which is $\psi \rightarrow \epsilon$. This specifies that if the precondition ψ is provable from the current state, then no message sending is required.

Compound protocols can be built up from these atomic protocols. If π_1 and π_2 are two protocols, then the following are also protocols: the protocol $\pi_1; \pi_2$, which represents sequential concatenation, such that π_1 is executed, followed by π_2; the protocol $\pi_1 \cup \pi_2$, which represents a choice between π_1 and π_2; and the protocol $\mathbf{var}_x^{\psi} \cdot \pi_1$, which is a protocol the same as π_1, except that a local variable x is available over the scope of π_1, but with the constraints ψ on x remaining unchanged throughout that scope. Any variable x already in the state is out of scope until π_1 finishes executing. In addition, $\mathcal{RASA}$ supports the referencing of protocols via their names. That is, for a protocol definition $N(x) \mathrel{\widehat{=}} \pi_1$, one can reference this from within another protocol using $N(y)$, where $y \in Var$.

A key feature of this language is that it has the same syntax and semantics at all dialogue levels. Single messages are themselves protocols, and the syntax and semantics for composing two atomic protocols is the same for composing two other composite protocols. Thus, individual utterances, sequences of utterances, protocols, and combinations of protocols can all be reasoned-over, modified, composed and invoked by agents participating in an interaction using the same reasoning mechanism.

Example 2.1. We present a small example of a simple interaction in which an agent, A, proposes that another agent, B, commits to P, and B can accept or refuse this proposal.

The semantics of $\mathcal{RASA}$ is compositional, so it makes sense to present the protocol in a bottom-up manner. First, we define the *Prop* protocol, an

atomic protocol which models A sending the proposal to B:

$$Prop(A, B, P) \mathrel{\widehat{=}} \text{true} \xrightarrow{c(A,B).propose(P)} prop(P)$$

The postcondition $prop(P)$ simply indicates that the current proposal is P. The Acc and Rej protocols model B accepting or rejecting the proposal respectively:

$$Acc(A, B) \mathrel{\widehat{=}} prop(P) \xrightarrow{c(B,A).accept(P)} cmt(B, A, P)$$
$$Rej(A, B) \mathrel{\widehat{=}} prop(P) \xrightarrow{c(B,A).reject(P)} \text{true}$$

The notation $cmt(B, A, P)$ is a constraint representing B's commitment to perform P for the creditor A.

Finally, we compose these three atomic protocols together into a composite protocol, which defines the order that the messages must occur:

$$Prot(A, B, P) \mathrel{\widehat{=}} Prop; (Acc \cup Ref)$$

This definition enforces the condition that the proposal must be sent by A before B can accept or reject it, and that B can only send either an accept or reject, but not both. In addition, if the path $Prop; Acc$ is taken, then B is committed to P.

One can see that, provided an agent understands the meaning of $cmt(B, A, P)$, such a protocol can be reasoned about at runtime. Firstly, agent A decides to use this protocol because it calculate that $cmt(B, A, P)$ is an outcome. If B agrees to using the protocol, then, after it receives the proposal, it can reason that accepting the proposal will lead to the state in which it is committed to performing B. If it does not accept, then there is no change, so it can decide its reply by analysing its goals and assessing their compatibility with the outcomes.

3 Illegal Protocols and Compositions

We define *composition* of a protocol as the process of either deriving a new atomic protocol, or taking one or more existing protocol definitions, and forming a larger compound protocol using the algebraic operators of the $\mathcal{RASA}$ specification language. For example, taking protocols π_1 and π_2, the composite protocol $\pi_1; \pi_2$ can be defined. The vision of first-class protocols includes agents composing their own protocols in this way, at runtime, if they do not have a protocol in their library that achieves their goals.

Clearly, there are verification issues regarding protocol composition. Some of these issues relate to the messages that are sent, the order that are sent in, and when they can be sent. Others are related to the rules and outcomes, such as whether they are correct with respect to an informal definition. Such issues are *domain specific*, and in fact, they may even be *agent specific*. In contrast, the purpose of this paper is to study *generic* properties of protocols. That is, issues regarding the relationship between protocols, and most specifically, the conditions under which they can be composed. Such properties would apply to every protocol, regardless of the domain.

In this section, we define and discuss one such generic property, which relates to protocol composition. We call this property *stuckness*. Stuckness is a property that should not be exhibited for any agent interaction protocol, and we assert that any protocol to be used should first be proved to be *stuckness free*.

Definition 3.1. Protocol Stuckness

The execution of a protocol becomes stuck if:

1. it is not terminated; and
2. it is either in a state from which no transition can be made, or the state is equivalent to false.

That is, at the current state, the rules of the protocol are such that no move can be made by any participant, or the current state contains an unsatisfiable constraint. We say that any protocol that can become stuck suffers from *stuckness*. Stuckness refers to some form of runtime error, in which agents can execute part of a protocol, but then come to a point at which they are unable to continue executing. Such a property is an undesirable property, and one which we want to prove is not possible for any protocol[1].

This is similar to the notion of deadlock in process algebras such as CSP [7]. However, stuckness does not occur as a result of processes waiting for each other, which is the definition of gridlock. Instead, the condition permitting an event to occur is a precondition, not another event, as it is in CSP.

Example 3.1. As an example of stuckness, consider the following protocol definition:

$$N(x) \mathrel{\widehat{=}} x = 1 \xrightarrow{c.a(x)} x = 2; x = 3 \xrightarrow{c.b(x)} x = 4$$

After $a(x)$ is sent across the channel, the state is $x = 2$. However, $x = 2$ does not entail the precondition $x = 3$, so the protocol execution becomes stuck.

To help us formally define stuckness for $\mathcal{RASA}$ protocol specifications, we first introduce some auxiliary definitions.

Definition 3.2. Local Constraints

Local constraints are the constraints on locally declared variables — that is, variables declared using the form $\mathbf{var}_x^{\psi} \cdot \pi$. Consider the following example protocol, which increments the values of x and y, provided that x is less than 10:

$$\mathbf{var}_{x_0,y_0}^{x_0=x \wedge y_0=y} \cdot (x < 10 \xrightarrow{c.a(x)} x = x_0 + 1 \wedge y = y_0 + 1)$$

If this is executed in the state $x = 0 \wedge y = 5$, then the local constraints over the scope of the atomic protocol would be $x_0 = 0 \wedge y_0 = 5$.

Throughout this section, when we discuss protocols, we will assume that they are evaluated under local constraints. If a protocol is not contained within the scope of a variable declaration, the local constraints are equivalent to the constraint 'true'. Further discussion of this is presented later in Section 4.1 at a point when their purpose should be clearer.

Definition 3.3. Weakest Explicit Precondition

The *weakest explicit precondition* of a protocol is the weakest (or most general) constraint that satisfies the precondition of the protocol. The weakest *explicit* precondition is in contrast to the weakest *calculated* precondition, which is the weakest constraint from which a protocol cannot become stuck. The terminology is used because the weakest calculated precondition must be computed taking into account the entire protocol, whereas the explicit precondition is taken as the disjunction of the preconditions of all of the atomic protocols that can be executed as the first step in the protocol. For example, consider the following protocol:

$$Q(x) \mathrel{\widehat{=}} x = 0 \xrightarrow{c.a(x)} x = 1; y = 0 \xrightarrow{c.b(x)} y = 1$$

[1] However, we note that there may be situations in which an agent desires an interaction to become stuck, e.g. to distract a competitor, as described in [6].

The weakest explicit precondition of this protocol is $x = 0$, because $x = 0$ is the most general constraint under which the protocol can begin to execute. However, the calculated precondition is $x = 0 \wedge y = 0$, because $y = 0$ must hold for the protocol to execute fully due to the fact that it is the precondition of the second sub-protocol, and that the first sub-protocol does not ensure this.

We define a function, $pre \in \pi \times \mathcal{L} \rightarrow \mathcal{L}$, which, for a protocol, π and local constraints, $\mathcal{L}$, returns the weakest *explicit* precondition of π. *pre* is defined formally as follows:

$$\begin{array}{lcl} pre(\psi \rightarrow \epsilon, \chi) & = & \psi \wedge \chi \\ pre(\psi \xrightarrow{c.\phi_m} \psi', \chi) & = & \psi \wedge \chi \\ pre(\pi_1; \pi_2, \chi) & = & pre(\pi_1, \chi) \\ pre(\pi_1 \cup \pi_2, \chi) & = & pre(\pi_1, \chi) \vee pre(\pi_2, \chi) \\ pre(\mathbf{var}_x^{\psi} \cdot \pi, \chi) & = & \exists_x pre(\pi, \exists_x \chi \wedge \psi) \\ pre(N(x), \chi) & = & pre(\pi[x/y], \chi) \\ & & \text{where } N(y) \mathrel{\widehat{=}} \pi \in D \end{array}$$

We discuss these definitions briefly. The explicit preconditions of an empty or atomic protocol is the explicit precondition, ψ, conjoined with the local constraints, χ. The explicit precondition of a sequential composition, $\pi_1; \pi_2$, is the precondition of π_1. For a choice, $\pi_1 \cup \pi_2$, it is the disjunction of the preconditions of π_1 and π_2. The explicit precondition of a variable declaration, $\mathbf{var}_x^{\psi} \cdot \pi$, is the precondition of π, but including the local constraints, ψ, and with x hidden from χ so the local declaration is not confused with any already-declared x. References to x in the precondition of π are then hidden, because x is local. Finally, the explicit precondition of a reference name protocol is the precondition of the corresponding definition with the parameters appropriately renamed. We omit the local constraints parameter if it is unnecessary.

Definition 3.4. Maximal Calculated Postcondition

The *maximal calculated postcondition* (or simply maximal postcondition) of a protocol is the most general constraint such that every end state satisfies that constraint, and and that it is satisfied only by those end states. We use the term *calculated* because, unlike the precondition, the calculated postcondition requires one to calculate the end state over entire traces of protocols.

The maximal calculated postcondition is defined as a function, $post \in (\pi \times \mathcal{L} \times \mathcal{L}) \rightarrow \mathcal{L}$, which, for a protocol, initial state, and local constraints, returns the constraint that satisfies all postconditions, and only those postconditions. This can be computed on a syntactic level using the following definitions:

$$\begin{array}{lcl} post(\psi \rightarrow \epsilon, \phi, \chi) & = & \phi \\ post(\psi \xrightarrow{c.\phi_m} \psi', \phi, \chi) & = & \exists_{free(\phi')} \phi \wedge \phi' \\ & & \text{where } \phi' \equiv \psi' \wedge \phi_m \wedge \chi \\ post(\pi_1; \pi_2, \phi, \chi) & = & post(\pi_2, post(\pi_1, \phi, \chi)) \\ post(\pi_1 \cup \pi_2, \phi, \chi) & = & post(\pi_1, \phi, \chi) \vee \\ & & post(\pi_2, \phi, \chi) \\ post(\mathbf{var}_x^{\psi} \cdot \pi, \phi, \chi) & = & \exists_x post(\pi, \exists_x \phi, \exists_x \chi \wedge \psi') \\ post(N(x), \phi, \chi) & = & post(\pi[x/y], \phi, \chi) \\ & & \text{where } N(y) \mathrel{\widehat{=}} \pi \in D \end{array}$$

We comment on this definition briefly. The maximal postcondition of an empty protocol executed from a pre-state is the pre-state, because it does not change. For an atomic protocol, $\psi \xrightarrow{c.\phi_m} \psi'$, the maximal postcondition is the weakest constraint, ϕ', that satisfies the explicit postcondition, ψ', the message ϕ_m, and the local constraints χ. This is clear from the definition in [9], in which the postcondition is equivalent to this, except it is constrained by ϕ'_m instead of ϕ_m, such that $\phi'_m \sqsupseteq \phi_m$, allowing agents to constrain messages. However, if we are calculating the most general constraint, then ϕ_m is the most general message constraint, so we use this rather than any further constrained message. In addition, we conjoin this constraint with the pre-state, ϕ, but with all free variables in ϕ' hidden from ϕ.

The maximal postcondition of the sequential composition $\pi_1; \pi_2$ under the state ϕ is the maximal postcondition of π_2 under the initial state $post(\pi_1, \phi, \chi)$. That is, the maximal postcondition from executing π_2 under the maximal postcondition of π_1 under ϕ. The maximal postcondition of the choice protocol $\pi_1 \cup \pi_2$ is the disjunction of the two protocol's maximal postconditions, because one of these postconditions will hold after executing the choice. The maximal postcondition of a variable declaration $\mathbf{var}_x^{\psi} \cdot \pi$ is the maximal precondition of π under the state $\exists_x \phi$ and with local constraints $\exists_x \chi \wedge \psi'$. References to x in ϕ and χ are hidden because any references already declared are out of scope. The maximal precondition of the referenced name $N(x)$ is the maximal precondition of the corresponding protocol π, with the parameters renamed. We omit the local constraints parameter if it is unnecessary.

Example 3.2. Refer to the definition of the protocol $Q(x)$ from above. The weakest precondition of the protocol is determined as follows:

$$\begin{array}{ll} & pre(Q(x)) \\ \equiv & pre(x = 0 \xrightarrow{c.a(x)} x = 1; y = 0 \xrightarrow{c.b(x)} y = 1) \\ \equiv & pre(x = 0 \xrightarrow{c.a(x)} x = 1) \\ \equiv & x = 0 \end{array}$$

The maximal postcondition is determined using the definition of *post*. The overall maximal postcondition of a protocol is the maximal postcondition with the weakest precondition as the initial state:

$$\begin{array}{ll} & post(Q(x), pre(Q(x))) \\ \equiv & post(Q(x), x = 0) \\ \equiv & post(x = 0 \xrightarrow{c.a(x)} x = 1; \\ & \qquad y = 0 \xrightarrow{c.b(y)} y = 1, x = 0) \\ \equiv & post(y = 0 \xrightarrow{c.b(y)} y = 1, \\ & \qquad post(x = 0 \xrightarrow{c.a(x)} x = 1), x = 0) \\ \equiv & post(y = 0 \xrightarrow{c.b(y)} y = 1, x = 1) \\ \equiv & y = 1 \wedge x = 1 \end{array}$$

Definition 3.5. Formal Definition of Stuckness

Definition 3.1 presented an informal definition of stuckness for a protocol. The notion of stuckness, and its related proof obligations, is similar to the way in which preconditions are verified in model-oriented specification languages such as Z [14] and B [13]. However, in these methods, the proof is generally of the form: $pre \rightarrow \exists post$. That is, the proof is that if the precondition holds, there exists at least one poststate that satisfies the postcondition.

This definition is not enough to prove that a protocol is stuckness free. Take for example, the following protocol:

$$\begin{array}{l} R(x) \mathrel{\widehat{=}} (x = 0 \xrightarrow{c.a(x)} x = 1 \ \cup \ x = 0 \xrightarrow{c.b(x)} x = 2); \\ \qquad\qquad x = 1 \xrightarrow{c.d(x)} x = 10 \end{array}$$

This defines a choice followed by an atomic protocol. The precondition for both options in the choice is

$x = 0$, therefore, the overall precondition is $x = 0$. If the protocol is in a state in which the precondition holds, then the left option can be chosen, resulting in the state $x = 1$. The precondition of the next protocol is $x = 1$, so execution can complete. However, if the right option is chosen, resulting in the state $x = 2$, then the protocol becomes stuck. So, even though the precondition of this protocol implies that there is a postcondition (take the first option of the choice), the protocol can still become stuck. Therefore, an alternate definition to the $pre \rightarrow \exists post$ definition is needed.

We formally define stuckness as a relation called *stuck*, in which, for a protocol π, an initial state ϕ, and local constraints χ, $stuck(\pi, \phi, \chi)$ is true if and only if π can become stuck from the starting state ϕ under the local constraints χ. Devising a neat, formal definition of stuckness for any arbitrary protocol, like the $pre \rightarrow \exists post$ from above, does not seem possible, so instead, we define it such that each protocol operator has its own definition, each shown in Table 1.

This definition is not immediately obvious, so we spend some time discussing it. The empty protocol, $\psi \rightarrow \epsilon$, can only become stuck if its precondition is not enabled. [9] specifies the denotational semantics of an atomic protocol, $\psi \xrightarrow{c.\phi_m} \psi'$, as follows[2], assuming that ϕ is the pre-state and χ the local constraints:

$$\exists \phi'_m, \phi' \bullet (\phi \wedge \chi \sqsupseteq \psi) \wedge (\phi'_m \sqsupseteq \phi_m) \wedge \phi' \equiv (\psi' \wedge \phi'_m \wedge \chi \wedge \exists_v \phi)$$
where v is the free variables in $\psi' \wedge \phi'_m \wedge \chi$

Here, ϕ'_m represents the message, and ϕ' the post-state. So, this definition says that $\phi \wedge \chi$ must satisfy the precondition, the message ϕ'_m must be a refinement of the message template ϕ_m, and the post-state is the postcondition, ψ', conjoined with the information from the message, ϕ'_m, conjoined with the local constraints, χ (which must hold after the execution), and finally, conjoined with $\exists_v \phi$, in which v is the set of variables that this protocol changed, therefore $\exists_v \phi$ is the information from the pre-state ϕ that is not overridden by the postcondition.

An atomic protocol is stuck under an initial state ϕ if it defines no behaviour. So, for the pre-state ϕ, this is equivalent to negating the definition from above:

$$\begin{aligned} & \neg \exists \phi'_m, \phi' \bullet (\phi \wedge \chi \sqsupseteq \psi) \wedge \\ & \quad (\phi'_m \sqsupseteq \phi_m) \wedge \phi' \equiv (\psi' \wedge \phi'_m \wedge \chi \wedge \exists_v \phi) \\ \equiv \quad & \forall \phi'_m, \phi' \bullet (\phi \wedge \chi \not\sqsupseteq \psi) \vee \phi'_m \not\sqsupseteq \phi_m \vee \\ & \quad \phi' \not\equiv (\psi' \wedge \phi'_m \wedge \chi \wedge \exists_v \phi) \end{aligned}$$

If for every ϕ'_m, $\phi'_m \not\sqsupseteq \phi_m$, that must mean that ϕ_m is equivalent to false, and therefore ϕ_m is unsatisfiable. Otherwise, if there is a ϕ'_m such that $\phi'_m \sqsupseteq \phi_m$, but for every ϕ', $\phi' \not\equiv (\psi' \wedge \phi'_m \wedge \chi \wedge \exists_v \phi)$, this must mean that $\psi' \wedge \phi'_m \wedge \chi \wedge \exists_v \phi$ is unsatisfiable. Therefore, the above predicate becomes:

$$\begin{aligned} & \forall \phi'_m \bullet (\phi \wedge \chi \not\sqsupseteq \psi) \vee \phi_m \sqsupseteq \text{false} \vee \\ & \quad (\phi'_m \sqsupseteq \phi_m \rightarrow (\psi' \wedge \phi'_m \wedge \chi \wedge \exists_v \phi) \sqsupseteq \text{false}) \end{aligned}$$

If for every ϕ'_m such that $\phi'_m \sqsupseteq \phi_m$ (including ϕ_m itself), $(\psi' \wedge \phi'_m \wedge \chi \wedge \exists_v \phi) \sqsupseteq$ false, then we can substitute ϕ_m for ϕ'_m in this predicate (removing the quantification), because ϕ_m is the most general case of ϕ'_m:

$$\begin{aligned} & (\phi \wedge \chi \not\sqsupseteq \psi) \vee \\ & \phi_m \sqsupseteq \text{false} \vee (\psi' \wedge \phi_m \wedge \chi \wedge \exists_v \phi) \sqsupseteq \text{false} \end{aligned}$$

Clearly, if $\phi_m \sqsupseteq$ false then also $(\psi' \wedge \phi_m \wedge \chi \wedge \exists_v \phi) \sqsupseteq$ false, so this first case is not necessary. Using the definition of implication, we are left with the following:

$$(\phi \wedge \chi \sqsupseteq \psi) \rightarrow (\psi' \wedge \phi_m \wedge \chi \wedge \exists_v \phi) \sqsupseteq \text{false}$$

However, $\exists_v \phi$ contains no information about the other variables in the post-state, because these variables are hidden by $\exists_v$, so conjoining it to the post-condition cannot result in an unsatisfiable constraint. We assume that ϕ is satisfiable (because it is the post-state of another protocol), we can remove it from the proof obligation. This leaves us with the following:

$$(\phi \wedge \chi \sqsupseteq \psi) \rightarrow (\psi' \wedge \phi_m \wedge \chi) \sqsupseteq \text{false}$$

Therefore, an atomic protocol becomes stuck if the precondition is not enabled by the current state, or if the postcondition (including any constraints specified by local variable declarations) and message are inconsistent with each other, which is an intuitive definition of stuckness.

The case for the sequentially composed protocol $\pi_1; \pi_2$ is straightforward; it is stuck under a pre-state ϕ if either π_1 is stuck under ϕ, or if, once π_1 has executed, π_2 is stuck under the end-state of π_1.

This case for a choice is less straightforward than it may first appear. An initial attempt to model stuckness for a choice protocol may be to specify that the protocol $\pi_1 \cup \pi_2$ is stuck under a state ϕ if and only if π_1 and π_2 are both stuck under ϕ. This is certainly true for the 'if', but not for the 'only if' — we are aiming to verify that protocol can never become stuck, so we have to consider the case in which one of the protocols is not stuck, so can execute to its end, but the other may be, so the composite protocol can still get stuck. An initial attempt to model this may be to specify that $\pi_1 \cup \pi_2$ is stuck in a state ϕ if and only if π_1 or π_2 is stuck in ϕ. However, this does not consider the case in which the precondition of only one protocol is enabled, which does not indicate a stuck protocol, because the other protocol may still be executed in another state that satisfies its precondition.

Considering all these cases, we have the following three cases in which the protocol $\pi_1 \cup \pi_2$ is stuck under ϕ:

- the precondition of the protocol is not enabled: $\phi \not\sqsupseteq pre(\pi_1 \cup \pi_2)$;
- the precondition of the protocol π_1 is enabled, but π_1 is stuck under ϕ: $\phi \sqsupseteq pre(\pi_1) \wedge stuck(\pi_1, \phi)$; and
- the precondition of the protocol π_2 is enabled, but π_2 is stuck under *phi*: $\phi \sqsupseteq pre(\pi_2) \wedge stuck(\pi_2, \phi)$.

Therefore, $stuck(\pi_1 \cup \pi_2, \phi)$ is true if and only if any of the above three conditions holds.

A variable declaration protocol, $\mathbf{var}_x^{\psi} \cdot \pi$, is stuck under ϕ if and only if the sub-protocol π is stuck under ϕ once the locals constraints on x are taken into consideration. This is straightforward to derive from the semantics from [9] — $\mathbf{var}_x^{\psi} \cdot \pi$ is stuck in state ϕ and local constraints χ if and only if π is stuck under the state $\exists_x \phi$ and local constraints $\exists_x \chi \wedge \psi$.

Finally, the referenced name $N(x)$ is stuck in ϕ if either N is not a name in the protocol specification, or if the protocol, π, corresponding to the definition of $N(y)$, becomes stuck in ϕ when variable y is renamed to x.

4 Verification of Protocols

In this section, we discuss how to prove that a protocol is stuckness free. Proofs are performed inductively

[2] The definition from [9] is in fact defined as a set of pre- and post-states, however, the definition presented here will suffice for discussion.

$stuck(\psi \rightarrow \epsilon, \phi, \chi)$	iff	$\phi \wedge \chi \not\sqsupseteq \psi$
$stuck(\psi \xrightarrow{c.\phi_m} \psi', \phi, \chi)$	iff	$(\phi \wedge \chi \sqsupseteq \psi) \rightarrow (\phi_m \wedge \psi' \wedge \chi \sqsupseteq \text{false})$
$stuck(\pi_1; \pi_2, \phi, \chi)$	iff	$stuck(\pi_1, \phi, \chi) \vee \exists \phi' \in \mathcal{L} \bullet \phi' \sqsupseteq post(\pi, \phi, \chi) \rightarrow stuck(\pi_2, \phi', \chi)$
$stuck(\pi_1 \cup \pi_2, \phi, \chi)$	iff	$\phi \not\sqsupseteq pre(\pi_1 \cup \pi_2, \chi) \vee$ $\phi \sqsupseteq pre(\pi_1, \chi) \wedge stuck(\pi_1, \phi, \chi) \vee$ $\phi \sqsupseteq pre(\pi_2, \chi) \wedge stuck(\pi_2, \phi, \chi)$
$stuck(\mathbf{var}_x^{\psi} \cdot \pi, \phi, \chi)$	iff	$stuck(\pi, \exists_x \phi, \exists_x \chi \wedge \psi)$
$stuck(D, N(x), \phi, \chi)$	iff	$N(y) \mathrel{\widehat{=}} \pi \notin D \vee stuck(\pi[x/y], \phi, \chi)$

Table 1: Formal Definition of Stuckness

over the structure of the protocols, and are divided into two cases: the first proving that there exists at least one state that satisfies the precondition of the protocol; and the second proving that for every state that does satisfy the precondition, executing the protocol from this state does not result in it becoming stuck.

4.1 Proof Obligations

The first proof obligation is showing that an initial state, other than false, exists. This is formally defined as follows:

$$\exists \phi \in \mathcal{L} \setminus \{\text{false}\} \bullet \phi \sqsupseteq pre(\pi, \chi)$$

Clearly, if there is no ϕ that satisfies the precondition of π, then the precondition is equivalent to false. Therefore, this proof obligation can be reduced to the following:

$$pre(\pi, \chi) \not\sqsupseteq \text{false}$$

This proof obligation is called the *initialisation proof*. The second case, proving that any states satisfying the precondition do not lead to the protocol to becoming stuck, is formally defined as follows:

$$\forall \phi \in \mathcal{L} \bullet \phi \in pre(\pi, \chi) \rightarrow \neg stuck(\pi, \phi, \chi)$$

Therefore, we are proving that any constraint that satisfies the precondition of a protocol cannot lead it to become stuck. However, this is equivalent to proving that π is not stuck for the most general case; that is, the weakest explicit precondition. Therefore, the following proof obligation will suffice:

$$\neg stuck(\pi, pre(\pi, \chi), \chi)$$

This proof obligation is called the *stuckness proof*.

In this section, we investigate methods for proving these properties for the different protocol operators. In addition, we make the assumption that structural induction is used to prove the legality of a protocol. Structural induction is a proof method in which one proves a property for the elements that make up a structure before proving the property for the structure itself. In the case of $\mathcal{RASA}$ protocols, the induction is performed over the protocol operators, with atomic protocols and the empty protocol as base cases. Our assumption seems reasonable given that the $\mathcal{RASA}$ framework is designed to encourage agents to compose protocols at runtime, and the protocols from which the new composite protocols are composed should already be verified as stuckness-free, so the inductive step will have already been performed.

However, proving that the sub-protocols are stuckness free does not necessarily prove it for the compound protocol. For example, consider the protocol $\pi_1; \pi_2$ — we have already established that all post-states of π_1 must satisfy the precondition of π_2, therefore, proving the legality of π_1 and π_2 is not enough to establish the legality of the composite protocol. In this section, we identify the additional proof obligations that must be discharged on top of the structural induction to prove that protocols are stuckness free.

Remark 1. Regarding Local Constraints

So far in this section, we have defined the weakest precondition and postcondition with respect to local constraints. It must be noted that we make some assumptions about the local constraints. Firstly, the reader may have noted that we define the weakest precondition of an atomic protocol, $\psi \xrightarrow{c.\phi_m} \psi'$, as $\psi \wedge \chi$, in which χ represents the constraints on local variables. However, we do not define from where χ is derived.

We assume that a protocol is proved correct only for the local constraints under which is used. Therefore, one must collect the local constraints as they inductively move down the structure of the protocol. For example, consider the following protocol, which increments the value of the variables x and y, provided $x < 10$ holds before hand:

$$\mathbf{var}_{x_0}^{x_0=x} \cdot \mathbf{var}_{y_0}^{y_0=y} \cdot x < 10 \xrightarrow{c.\phi_m} x = x_0{+}1 \wedge y = y_0{+}1$$

To evaluate the precondition of this protocol, one would assume that there are no local constraints. Then, the inner variable declaration, $\mathbf{var}_{y_0}^{y_0=y} \cdot \ldots$ is evaluate under the local constraints $x_0 = x$, and the inner atomic protocol is evaluated under the local constraints $x_0 = x \wedge y_0 = y$. Therefore, the weakest explicit precondition of the inner atomic protocol is the precondition, $x < 10$, conjoined with the local constraints, to get $x < 10 \wedge x_0 = x \wedge y_0 = y$, which simplifies to $x < 10 \wedge x_0 < 10 \wedge y_0 = y$.

4.1.1 Empty Protocol

From the definitions in Section 3, we know that the empty protocol can only become stuck if its precondition does not hold. For the initialisation proof, we substitute in an empty protocol for π from the generic proof obligation and simplify:

$$\begin{array}{ll} & pre(\psi \rightarrow \epsilon, \chi) \not\sqsupseteq \text{false} \\ \equiv & \psi \wedge \chi \not\sqsupseteq \text{false} \qquad \text{from defn. of } pre \end{array}$$

Therefore, to discharge the initialisation proof for an empty protocol, one must simply prove that its weakest precondition, $\psi \wedge \chi$ in this instance, is satisfiable.

To prove the absence of stuckness in a protocol, we substitute in an empty protocol for π from the generic proof obligation and simplify:

$$\begin{array}{ll} & \neg stuck(\psi \rightarrow \epsilon, pre(\psi \rightarrow \epsilon, \chi), \chi) \\ \equiv & \neg(\psi \wedge \chi \not\sqsupseteq \psi) \quad \text{from defn. of } stuck \\ \equiv & \psi \wedge \chi \sqsupseteq \psi \qquad \text{double negation} \end{array}$$

This final line is trivially true, so we conclude that no proof obligation is necessary for the stuckness proof.

4.1.2 Atomic Protocols

Atomic protocols, along with the empty protocol, are the base cases of the inductive proof — that is, there are no sub-protocols that we must verify before proving this protocol is legal.

For the initialisation proof of atomic protocols, we substitute in an atomic protocol for π from the generic proof obligation and simplify:

$$\begin{aligned} & pre(\psi \xrightarrow{c.\phi_m} \psi', \chi) \not\sqsupseteq \text{false} \\ \equiv\ & \psi \wedge \chi \not\sqsupseteq \text{false} \qquad \text{from defn. of } pre \end{aligned}$$

Therefore, to discharge the initialisation proof for an atomic protocol, one must simply prove that its weakest precondition, $\psi \wedge \chi$ in this instance, is satisfiable.

To prove the absence of stuckness in a protocol, we substitute in an atomic protocol for π from the generic proof obligation and simplify:

$$\begin{aligned} & \neg stuck(\psi \xrightarrow{c.\phi_m} \psi', pre(\psi \xrightarrow{c.\phi_m} \psi', \chi), \chi) \\ \equiv\ & \neg stuck(\psi \xrightarrow{c.\phi_m} \psi', \psi \wedge \chi, \chi) \\ & \text{from defn. of } pre \\ \equiv\ & \neg(\psi \wedge \chi \sqsupseteq \psi \rightarrow (\psi' \wedge \phi_m \wedge \chi \sqsupseteq \text{false})) \\ & \text{from defn. of } stuck \\ \equiv\ & \psi \wedge \chi \not\sqsupseteq \psi \vee (\psi' \wedge \phi_m \wedge \chi) \not\sqsupseteq \text{false} \\ & \text{from de Morgan's laws} \\ \equiv\ & (\psi' \wedge \phi_m \wedge \chi) \not\sqsupseteq \text{false} \\ & \text{because } \psi \wedge \chi \sqsupseteq \psi \text{ for any } \psi, \chi \end{aligned}$$

Therefore, to discharge the stuckness proof obligation, one must prove that there exists a postcondition from the weakest precondition, by proving that $\psi' \wedge \phi_m \wedge \chi$ is satisfiable.

4.1.3 Sequential Composition

For the initialisation proof of sequential composition protocols, we substitute in a sequential composition for π from the generic proof obligation and simplify:

$$\begin{aligned} & pre(\pi_1; \pi_2) \not\sqsupseteq \text{false} \\ \equiv\ & pre(\pi_1) \not\sqsupseteq \text{false} \qquad \text{from defn. of } pre \end{aligned}$$

Therefore, to discharge the initialisation proof obligation for a sequential composition, $\pi_1; \pi_2$, one must discharge the proof obligation for π_1. Assuming an inductive proof over the structure of $\pi_1; \pi_2$, this will have already been discharged, therefore, such a proof is not necessary.

To prove the absence of stuckness in a protocol, we substitute in a sequential composition for π from the generic proof obligation and simplify:

$$\begin{aligned} & \neg stuck(\pi_1; \pi_2,\ pre(\pi_1; \pi_2,\ \chi),\ \chi) \\ \equiv\ & \neg stuck(\pi_1; \pi_2, pre(\pi_1, \chi),\ \chi) \\ & \text{from defn. of } pre \\ \equiv\ & \neg(stuck(\pi_1, pre(\pi_1, \chi),\ \chi) \vee \\ & \quad stuck(\pi_2, post(\pi_1, pre(\pi_1, \chi), \chi), \chi) \\ & \text{from defn. of } stuck \\ \equiv\ & \neg stuck(\pi_1, pre(\pi_1, \chi),\ \chi) \wedge \\ & \neg stuck(\pi_2, post(\pi_1, pre(\pi_1, \chi), \chi), \chi) \\ & \text{from de Morgan's laws} \end{aligned}$$

Assuming an inductive proof over the structure of $\pi_1; \pi_2$, the stuckness proof obligations for π_1 and π_2 would have already been discharged. Therefore, $\neg stuck(\pi_1, pre(\pi_1, \chi),\ \chi)$ need not be proved again. However, protocol π_2 would have been proved to be stuckness free only for its precondition, so we need to prove that every postcondition of π_1 satisfies the precondition of π_2. We can do this by proving that the maximal postcondition of π_1, $post(\pi_1, pre(\pi_1, \chi)$, is a stronger constraint than the precondition of π_2:

$$post(\pi_1, pre(\pi_1, \chi), \chi) \sqsupseteq pre(\pi_2, \chi)$$

Therefore, to prove that sequentially composed protocol, $\pi_1; \pi_2$, is stuckness free, one must prove that the postcondition of π_1, under its weakest precondition, implies weakest precondition of π_2.

We also include an additional proof obligation for special case for sequential composition: protocols of the form $\pi_1; (\pi_2 \cup \pi_3)$. In such protocols, it is possible that the protocol is not stuck, however, that one of π_2 or π_3 is never enabled. For example:

$$\begin{aligned} N(x) \mathrel{\widehat{=}} & x = 1 \xrightarrow{c.a(x)} x = 2; \\ & (x = 2 \xrightarrow{c.b(x)} x = 3 \cup x \neq 2 \xrightarrow{c.d(x)} x = 4) \end{aligned}$$

In this example, the postcondition $x = 2$ enables the left side of the choice every time, but never the right hand side. While this does not imply stuckness, because the left case is always enabled, it does mean that the right hand side is unnecessary. Therefore, in this case, we prove that $\pi_1; \pi_2$ and $\pi_1; \pi_3$ are both stuckness free. This seems reasonable, because

$$\pi_1; (\pi_2 \cup \pi_3) \quad = \quad \pi_1; \pi_2 \cup \pi_1; \pi_3$$

for any π_1, π_2, and π_3, therefore, it would be inconsistent for the protocol on the right side of the equality to be declared stuck, while the protocol on the left side is not. Similarly protocols of the format $(\pi_1 \cup \pi_2); \pi_3$ can be broken into the two cases $\pi_1; \pi_3$ and $\pi_2; \pi_3$. This is strictly not necessary, but it may make a proof of such a property more straightforward.

4.1.4 Choice

For the initialisation proof of a choice protocol, we substitute in a choice for π from the generic proof obligation and simplify:

$$\begin{aligned} & pre(\pi_1 \cup \pi_2) \not\sqsupseteq \text{false} \\ \equiv\ & pre(\pi_1) \vee pre(\pi_2) \not\sqsupseteq \text{false} \end{aligned}$$

Therefore, to discharge the initialisation proof obligation for a choice, $\pi_1 \cup \pi_2$, one must prove that one of π_1 or π_2 has an initialisation state. Assuming an inductive proof over the structure of $\pi_1 \cup \pi_2$, this will have already been discharged for both π_1 and π_2, therefore, it trivially holds for either, and this proof is not necessary.

To prove the absence of stuckness in a choice protocol, we substitute in a choice for π from the generic proof obligation and simplify:

$$\begin{aligned} & \neg stuck(\pi_1 \cup \pi_2,\ pre(\pi_1 \cup \pi_2),\ \chi) \\ \equiv\ & \neg stuck(\pi_1 \cup \pi_2,\ pre(\pi_1) \vee pre(\pi_2),\ \chi) \\ & \text{from defn. of } pre \\ \equiv\ & \neg\big((pre(\pi_1) \vee pre(\pi_2) \not\sqsupseteq pre(\pi_1) \vee pre(\pi_2)) \vee \\ & \quad pre(\pi_1) \vee pre(\pi_2) \sqsupseteq pre(\pi_1) \wedge \\ & \quad\quad stuck(\pi_1, pre(\pi_1) \vee pre(\pi_2), \chi) \vee \\ & \quad pre(\pi_2) \vee pre(\pi_2) \sqsupseteq pre(\pi_2) \wedge \\ & \quad\quad stuck(\pi_2, pre(\pi_1) \vee pre(\pi_2), \chi)\big) \\ & \text{from defn. of } stuck \end{aligned}$$

The first line of the above predicate is trivially false, because $pre(\pi_1) \vee pre(\pi_2)$ enables one of the preconditions of π_1 and π_2 respectively. In addition, if we are proving that π_1 is not stuck in $pre(\pi_1) \vee pre(\pi_2)$ in which $pre(\pi_1) \vee pre(\pi_2) \sqsupseteq pre(\pi_1)$, then we can prove this for the more general case of $pre(\pi_1)$, and similarly for π_2. So, we are left with the following:

$$\begin{aligned} \Leftarrow\ & \neg\big(stuck(\pi_1, pre(\pi_1), \chi) \vee stuck(\pi_2, pre(\pi_2), \chi)\big) \\ \equiv\ & \neg stuck(\pi_1, pre(\pi_1), \chi) \wedge \neg stuck(\pi_2, pre(\pi_2), \chi) \end{aligned}$$

So, we are left to prove that protocols π_1 and π_2 are not stuck from their weakest precondition. Assuming an inductive proof over the structure of $\pi_1 \cup \pi_2$, π_1 and π_2 would have already been proven to not become stuck from their weakest precondition. This implies that the composing two stuckness-free protocols with the choice operator will produce a new protocol that is stuckness free, and therefore, no proof obligation is necessary.

4.1.5 Variable Declaration

For the initialisation proof of variable declarations, we substitute in a variable declaration for π from the generic proof obligation and simplify:

$$\begin{array}{ll} & pre(\mathbf{var}_x^{\psi} \cdot \pi) \not\supseteq \text{false} \\ \equiv & \exists_x pre(\pi, \exists_x \chi \wedge \psi) \not\supseteq \text{false} \quad \text{from defn. of } pre \\ \equiv & pre(\pi, \exists_x \chi \wedge \psi) \not\supseteq \text{false} \quad \text{from defn. of } \exists \end{array}$$

Assuming an inductive proof over the structure of $\mathbf{var}_x^{\psi} \cdot \pi$, this will have already been proved for π, therefore, no additional proof obligation is necessary. However, if we assume an agent is composing an existing protocol π that has been verified, but not under the local constraints $\exists_x \chi \wedge \psi$, then this additional proof obligation must be discharged. This can be done by either proving the above, or by proving that $\exists_x \chi \wedge \psi$ entails the local constraints under which it was previously proved. The latter case is possible because, if this is proved for a specific case, then it will also hold for a more general case.

To prove the absence of stuckness in a variable declaration, we substitute in a variable declaration for π from the proof obligation and simplify:

$$\begin{array}{ll} & \neg stuck(\mathbf{var}_x^{\psi} \cdot \pi,\ pre(\mathbf{var}_x^{\psi} \cdot \pi,\ \chi),\ \chi) \\ \equiv & \neg stuck(\mathbf{var}_x^{\psi} \cdot \pi,\ \exists_x pre(\pi, \exists_x \chi \wedge \psi),\ \chi) \\ & \text{from defn. of } pre \\ \equiv & \neg stuck(\pi,\ \exists_x pre(\pi, \exists_x \chi \wedge \psi), \exists_x \chi \wedge \psi) \\ & \text{from defn. of } stuck \end{array}$$

Assuming an inductive proof over the structure of $\mathbf{var}_x^{\psi} \cdot \pi$, the following would already have been proved:

$$\neg stuck(\pi,\ pre(\pi, \exists_x \chi \wedge \psi), \exists_x \chi \wedge \psi)$$

This looks close to the predicate we wish to prove, except that the weakest precondition of π is $pre(\pi, \exists_x \chi \wedge \psi)$, rather than $\exists_x pre(\pi, \exists_x \chi \wedge \psi)$, which is even weaker because x is hidden, and therefore unconstrained. However, these predicates are in fact the same, because the constraints on x specified by ψ in $\exists_x pre(\pi, \exists_x \chi \wedge \psi)$ may be hidden, but they still influence the behaviour of π via the ψ in the local constraints. Therefore, if any value of x from ψ is causing π to become stuck, the same value will also cause this to become stuck from ψ. This means that a stuckness proof obligation is not necessary for variable declarations. This is not difficult to see considering that π is already proved to be never become stuck when its precondition is enabled, and the precondition of $\mathbf{var}_x^{\psi} \cdot \pi$ is exactly those states that, when taking into consideration the additional variable x and its constraints, satisfy the precondition of π.

However, as with the initialisation proof, if we assume that π has been proved in the context of a different set of local constraints, then the above proof obligation must be discharged for the local constraints $\exists_x \chi \wedge \psi$.

4.1.6 Reference Name

For the initialisation proof of reference names, we substitute in a reference name for π from the generic proof obligation and simplify:

$$\begin{array}{ll} & pre(N(x), \chi) \not\supseteq \text{false} \\ \equiv & pre(\pi[x/y], \chi) \not\supseteq \text{false} \quad \text{where } N(y) \mathrel{\widehat{=}} \pi \in D \\ \equiv & pre(\pi, \chi) \not\supseteq \text{false} \\ & \text{because } \phi \in \mathcal{L} \text{ implies } \phi[y/x] \in \mathcal{L} \end{array}$$

Therefore, to discharge the initialisation proof obligation for a reference name, $N(x)$, in which $N(y) \mathrel{\widehat{=}} \pi$ is in the protocol specification D, one must prove that there exists a pre-state that satisfies the precondition of π. Assuming an inductive proof over the structure of the protocol, π will have already been proven to have an initial state, therefore, this proof obligation is not necessary in the case that $N(y) \mathrel{\widehat{=}} \pi$.

However, we have a different case, in which $N(y)$ is not a name in the protocol specification:

$$\begin{array}{ll} & pre(N(x)) \not\supseteq \text{false} \\ \equiv & \text{nothing} \quad \text{where } N(y) \mathrel{\widehat{=}} \pi \notin D \end{array}$$

Therefore, to discharge the initialisation proof obligation for a reference name, $N(x)$, one has to prove that $N(y) \mathrel{\widehat{=}} \pi$ is a named protocol in the protocol specification. That is, one must prove:

$$N(y) \mathrel{\widehat{=}} \pi \in D \quad \text{for some } y$$

To prove the absence of stuckness in a reference name protocol, we substitute in a reference name for π from the proof obligation and simplify:

$$\begin{array}{ll} & \neg stuck(N(x), pre(N(x), \chi), \chi) \\ \equiv & N(y) \mathrel{\widehat{=}} \pi \in D \wedge \neg stuck(\pi[x/y], pre(\pi[x/y]), \chi)) \end{array}$$

Clearly, the right side of the conjunction is equivalent to the proof obligation of π, so assuming an inductive proof over the structure of the protocol, this part of the proof obligation is not necessary. The left side of the conjunction has already been proved for the initialisation proof of this protocol, therefore, no additional proof obligation is necessary.

4.2 Proving the Absence of Stuckness

When composing a protocol, we want to prove that our compositions are legal. To do this, we can discharge the proof obligations outlined in this section. However, many of these proof obligations refer to the local constraints, so one must relate these to the composition we are verifying. In this section, we outline an algorithm for verifying an arbitrary protocol.

The algorithm, which we call *prove*, takes a protocol and local constraints, and returns true or false, indicating whether the composition is legal or not. The sketch is shown in Figure 1, and should be clear to understand from the discussions in this section.

To prove a protocol π, is legal, one simply uses $prove(\pi, \text{true})$, in which true is the local constraint; that is, there are no local variables, so there are no local constraints.

The reader may have already noted a problem with the algorithm in Figure 1 regarding termination. If there are mutually recursive references to names in a protocol, this algorithm will not terminate. For example, take the following protocol specification:

$$\begin{array}{lll} A & \mathrel{\widehat{=}} & B \cup \epsilon \\ B & \mathrel{\widehat{=}} & A \cup \epsilon \end{array}$$

This protocol can run infinitely, as well as terminate (if an agent chooses ϵ). The *prove* algorithm, as it is above, will attempt to prove the legality of A by looking up its respective definition, $B \cup \epsilon$ and

$prove(\pi, \chi) : \{\text{true}, \text{false}\}$
  if $\pi = \psi \rightarrow \epsilon$ then
    return $\psi \wedge \chi \not\sqsupseteq \text{false}$
  else if $\pi = \psi \xrightarrow{c.\phi_m} \psi'$ then
    return $\psi \wedge \chi \not\sqsupseteq \text{false}$ and $\psi \wedge \phi_m \wedge \chi \not\sqsupseteq \text{false}$
  else if $\pi = \pi_1; (\pi_2 \cup \pi_3)$ then
    return $prove(\pi_1; \pi_2, \chi)$ and $prove(\pi_1; \pi_3, \chi)$
  else if $\pi = \pi_1; \pi_2$ then
    return $prove(\pi_1, \chi)$ and $prove(\pi_2, \chi)$ and
      $post(\pi_1, pre(\pi_1), \chi) \sqsupseteq pre(\pi_2, \chi)$
  else if $\pi = \pi_1 \cup \pi_2$ then
    return $prove(\pi_1, \chi)$ and $prove(\pi_2, \chi)$
  else if $\pi = \mathbf{var}_x^{\psi} \cdot \pi_1$ then
    return $prove(\pi_1, \exists_x \chi \wedge \psi)$
  else if $\pi = N(x)$ then
    return $N(y) \mathrel{\widehat{=}} \pi \ \in \ D$ and $prove(\pi[x/y], \chi)$

Figure 1: Algorithm for Proving Protocols are Legal

$compose(\pi) : \{\text{true}, \text{false}\}$
  if $\pi = \psi \rightarrow \epsilon$ then
    return $\psi \not\sqsupseteq \text{false}$
  else if $\pi = \psi \xrightarrow{c.\phi_m} \psi'$ then
    return $\psi \not\sqsupseteq \text{false}$ and $\psi \wedge \phi_m \not\sqsupseteq \text{false}$
  else if $\pi = \pi_1; (\pi_2 \cup \pi_3)$ then
    return $compose(\pi_1; \pi_2, \chi)$ and
      $compose(\pi_1; \pi_3, \chi)$
  else if $\pi = \pi_1; \pi_2$ then
    return $post(\pi_1, pre(\pi_1), \text{true}) \sqsupseteq pre(\pi_2, \text{true})$
  else if $\pi = \mathbf{var}_x^{\psi} \cdot \pi_1$ then
    return $prove(\pi_1, \psi)$
  else if $\pi = N(x)$ then
    return $N(y) \mathrel{\widehat{=}} \pi \ \in \ D$
  else return true

Figure 2: Algorithm for Proving Compositions are Legal

verifying that. B will in turn be verified by looking up its definition, $A \cup \epsilon$, and verifying that, and the problem starts again with A, therefore running infinitely. A minor adjustment to the above algorithm that keeps track of the named protocols and parameters that have been verified will resolve this problem.

4.3 Proving Compositions

If we are to compose two existing protocols, then the proof system is somewhat different to that outlined in the *prove* algorithm. This is because we assume that, if an agent is to construct a new protocol from existing protocols that reside in some form of library, then the protocols in the library are already legal. This means we only have to discharge the proof obligations outlined in Section 4.1. Therefore, a composition algorithm only has to prove the following:

- For every empty protocol:
  - prove the existence of an initial state.
- For every atomic protocol:
  - prove the existence of an initial state; and
  - prove the existence of a post-state.
- For every sequential composition:
  - prove that every post-state of the protocol on the left enables a precondition on the right.
- For every variable declaration:
  - prove that adding the constraints on the new local variable do not lead the sub-protocol becoming stuck.
- For every referenced name:
  - prove that the name is in the protocol specification.

Proving the legality of compositions can be done with less effort than proving the legality of entire protocols. In addition, if the protocols are taken from the library, then they will have been proved under the assumption that there are no local constraints. A composition is not under any local constraints either, so we know that the sub-protocols used in the composition are proved under the same local constraints. The exception to this is variable declarations. If we compose protocol π with a declaration of a variable, such as $\mathbf{var}_x^{\psi} \cdot \pi$, then we must prove, using the *prove* algorithm from Figure 1 that π is valid under the local constraints ψ. The reason for this is easily demonstrated with an example. The atomic protocol, $x < 10 \xrightarrow{c.\phi_m} x = y + 1$, which is a legal protocol, can be composed with the variable declaration $\mathbf{var}_{x,y}^{x=y} \cdot x < 10 \xrightarrow{c.\phi_m} x = y + 1$. This new composition is not legal, because the local constraint $x = y$ must hold over the scope of x and y, but this makes the postcondition unsatisfiable: there does not exist values for x and y such that $x = y$ and $x = y + 1$. Therefore, in the case of creating a variable declaration composed from a legal protocol, one must prove that this protocol is satisfied under the specified local constraints.

The procedure for verifying composite protocols that have been derived from other legal protocols is specified in the algorithm called *compose*, shown in Figure 2.

Note that *compose* does not take any local constraints as a parameter. As well as not recursively proving the legality of the sub-protocols, *compose* is also different to *prove* in that atomic protocols are verified without the local constraints χ, because the local constraints are assume to be just 'true'. The variable declaration case uses the *prove* algorithm from Figure 1, for the reasons discussed above. Finally, any instances of choice protocols need not be verified, because they are legal by default, therefore, these default to the final 'else', which returns 'true'.

5 Related Work

There are a handful of languages that have been used for first-class protocol specification. Various authors have had success with approaches based on Petri Nets [3] and on declarative specification languages [4, 5, 15], as well as an algebraic language similar to $\mathcal{RASA}$ called the Lightweight Coordination Calculus [12]. [8] presents a detailed comparison of these languages, including $\mathcal{RASA}$, so we do not cover this here. The authors of the cited work have discussed and demonstrated proof methods for these languages, but these involved proving domain-specific properties, not generic properties such as stuckness.

As noted in Section 3, verification of model-oriented specifications often prove generic properties that resemble our work. For example, the Cogito [1] and B method [13] development architectures both recommend an initialisation proof, which is similar our notion of an initialisation proof, and that a precondition of an operation should satisfy its postcon-

dition, which is similar to the second proof obligation for atomic protocols in $\mathcal{RASA}$. However, these methods differ significantly because operations in these languages as modelled as relations between precondition and postconditions using predicates, whereas $\mathcal{RASA}$ specifies no relationship between precondition and postconditions, just some information that holds after a protocol executes.

Our notion of stuckness is similar to that of Pierce's definition of stuckness for programs [11], and in fact, this is from where we take the term. However, Pierce's motivation for identifying stuckness is to define programming languages such that it is not possible to write a program that becomes stuck, whereas we are aiming to prove the absence of stuckness, while defining the $\mathcal{RASA}$ language such that stuckness is possible. Pierce uses *runtime exceptions* to handle programs that would otherwise become stuck. Exceptions could be added to $\mathcal{RASA}$, but we feel that raising runtime exceptions when a protocol cannot continue executing is no more desirable than becoming stuck, so we prefer to eliminate stuckness for protocols before their use.

6 Conclusions and Future Work

By treating agent interaction protocols as first-class entities, $\mathcal{RASA}$ permits protocols to be dynamically inspected, referenced, invoked, composed, and shared by ever-changing collections of agents engaged in interaction. The task of protocol composition, selection, and invocation may thus be undertaken by agents rather than agent designers, acting at run-time rather than at design-time. Frameworks such as this will be necessary to achieve the full vision of collections of intelligent autonomous agents interacting in dynamic environments.

This paper takes us one step towards such visions, by identifying generic cases in which first-class protocols are deemed to be illegal. We define a notion of *stuckness* for agent interaction protocols, and formally defined stuckness for the $\mathcal{RASA}$ framework. We have presented a method for proving that a protocol composition is legal, and noted the specific proof obligations that must be discharged when composing legal protocols into larger protocols. Proof obligations are specific to the composition operator that is being used in the composition, and can be discharged by the agents at runtime. Emphasis is placed on protocols specified in the $\mathcal{RASA}$ protocol language, but such ideas would be applicable to protocols specified in any language with features similar to $\mathcal{RASA}$'s.

Before our full visions are realised, significant further work is required. In other work [9], we are investigating methods for documenting the outcomes of protocols, which will allow agents to search for the protocols that best achieve their goals. In addition, meta-protocols are needed that allow agents to propose and negotiate which protocols are to be used, and suitable protocols for doing so will be investigated. To develop and test these ideas, we plan a prototype implementation in which agents negotiate the exchange of information using protocols specified using the $\mathcal{RASA}$ framework.

Acknowledgements

We are grateful for financial support from the EC-funded PIPS project (EC-FP6-IST-507019), website: `http://www.pips.eu.org`, the EC-funded ASPIC project (IST-FPC-002307), website: `http://www.argumentation.org/`, and the EPSRC Market-Based Control project (GR/T10657/01), website: `http://www.marketbasedcontrol.com/`.

References

[1] A. Bloesch and O. Traynor. The Cogito tool architecture. Technical Report 95-7, Software Verification Research Centre, 1995.

[2] F. S. De Boer, M. Gabbrielli, E. Marchiori, and C. Palamidessi. Proving concurrent constraint programs correct. *ACM Transactions on Programming Languages and Systems*, 19(5):685–725, September 1997.

[3] L. P. de Silva, M. Winikoff, and W. Liu. Extending agents by transmitting protocols in open systems. In *Proceedings of the Challenges in Open Agent Systems Workshop*, Melbourne, Australia, 2003.

[4] N. Desai, A. U. Mallya, A. K. Chopra, and M. P. Singh. OWL-P: A methodology for business process modeling and enactment. In *Workshop on Agent Oriented Information Systems*, pages 50–57, July 2005.

[5] N. Desai and M. P. Singh. A modular action description language for protocol composition. In *Proceedings of the 22nd Conference on Artificial Intelligence (AAAI)*, pages 962–967. AAAI Press, 2007.

[6] P. E. Dunne. Prevarication in dispute protocols. In G. Sartor, editor, *Proceedings of the Ninth International Conference on AI and Law*, pages 12–21, New York, NY, USA, 2003. ACM Press.

[7] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, 1985.

[8] J. McGinnis and T. Miller. Amongst first-class protocols. In *Engineering Societies in the Agents World VIII*, LNAI, 2007. (To Appear).

[9] T. Miller and P. McBurney. Executable logic for reasoning and annotation of first-class interaction protocols. Technical Report ULCS-07-015, University of Liverpool, Department of Computer Science, 2007.

[10] T. Miller and P. McBurney. Using constraints and process algebra for specification of first-class agent interaction protocols. In G. O'Hare, A. Ricci, M. O'Grady, and O. Dikenelli, editors, *Engineering Societies in the Agents World VII*, volume 4457 of *LNAI*, pages 245–264, 2007.

[11] B. C. Pierce. *Types and programming languages*. MIT Press, Cambridge, MA, USA, 2002.

[12] D. Robertson. Multi-agent coordination as distributed logic programming. In *Proceedings of the International Conference on Logic Programming*, volume 3132 of *LNCS*, pages 416–430. Springer, 2004.

[13] S. Schneider. *The B-Method: An Introduction*. Palgrave, 2001.

[14] J. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, 2nd edition, 1992.

[15] P. Yolum and M. P. Singh. Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. *Annals of Mathematics and AI*, 42(1–3):227–253, 2004.

An Investigation of the State Formation and Transition Limitations for Prediction Problems in Recurrent Neural Networks

Angel Kennedy and Cara MacNish
School of Computer Science and Software Engineering
The University of Western Australia
{angel,cara}@csse.uwa.edu.au

Abstract

Recurrent neural networks are able to store information about previous as well as current inputs. This "memory" allows them to solve temporal problems such as language recognition and sequence prediction, and provide memory elements for larger cognitive networks. It is generally understood that there is an (increasing) relationship between the number of nodes (and connections) in a network, the capabilities of the network, and the amount of training required. However the specifics of this relationship are less well understood. In particular, given that the state of a recurrent network is encoded as a real-valued vector of activation levels, even for small networks there are infinitely many states to choose from. What then determines, or limits, the capabilities of the network?

In this paper we use dynamical systems techniques to examine this question in regard to temporal lag. We show that for simple delay problems that the network is unable to solve, the system is able to learn sufficient state representations, but appears to be unable to create transitions that allow it to access those states in the correct order (or equivalently, is unable to arrange its states to suit the transitions that it can support).

1 Introduction

Recurrent neural networks (RNNs) are able to store information about previous as well as current inputs. This "memory" allows them to solve temporal problems such as language recognition and sequence prediction [1],[2], and provide memory elements for larger cognitive networks, such as *Long Short-Term Memory* (LSTM) [3]. It is generally understood that there is an (increasing) relationship between the number of nodes (and connections) in a network, the capabilities of the network, and the amount of training required. However the specifics of this relationship are less well understood.

In our work we are interested in the capabilities of recurrent networks as memory units in the context of larger cognitive systems. More specifically we are interested in the impact of changing the activation dynamics or altering the amount of structure in the network on these capabilities. One important aspect of this is the degree to which recurrent networks can learn to "remember" across time lags. It has been argued, for example, that the error in RNNs trained by backpropagation either vanishes or blows up causing RNNs to fail in the presence of time lags greater than 5-10 discrete time steps between input and corresponding output events [3],[4]. However these papers do not examine the relationship between network size (and hence "degrees of freedom") and time lag.

In the last decade or so a number of authors have attempted to elucidate the "inner workings" of RNNs (see for example [1],[5] and [6]). Many of these papers focussed on the ability observed in RNNs to perform sequential recognition tasks, such as recognising regular languages, and thereby mimic finite state machines (FSMs). In his seminal paper, Casey [1] presented a number of theoretical results linking the dynamics of RNNs and their performance on FSM computations. He showed, for example, that for an RNN to perform FSM behaviour it must necessarily partition its state space into disjoint regions that correspond to the states of the minimal deterministic finite automata (DFA). He also showed that cycles in the DFA will induce attractors in transition maps of the dynamical system induced by the network. We will explain these concepts in more detail in Section 2.

A number of authors have more recently determined computational limitations for a variety of RNNs (e.g. Hopfield, threshold, sigmoid) in modelling DFAs in terms of the relationship between the number of states of the deterministic finite automata that have to be modelled and the number of nodes or units a network requires (see [7] for a review). Whilst useful, these analyses suffer from several drawbacks in terms of providing the level of understanding we require. One is that they define the complexity only for the case where the network robustly models the underlying DFA. In our current work we do not necessarily require one hundred percent accuracy from the network. Another problem is that they do not seem to address the question in a way that considers qualities of the required transitions of the underlying DFA. Preliminary analyses by Casey [1] suggested that particular qualities of the transitions or the relationship between transitions within the DFA are important for the required complexity of the underlying dynamical system, in our case the RNN.

In this paper we apply the dynamical systems techniques used by Casey and others to examine the practical problem of time lag in RNNs, and its relationship to network size. To achieve this we run experiments on small RNNs that attempt to learn to delay a binary signal. When the number of nodes is equal to or larger than the number of time delay steps these problems are easily solved by "pipelining" the units, and this is indeed what we found the RNNs learned to do. The more interesting case is where the number of units is less than the required delay. In this case the ability of the network to solve the problem, for a fixed difference between number of nodes and delay length, depends on the number of nodes. In partic-

ular, smaller networks are less successful at compensating for a fixed difference.

Intuitively one might expect a smaller network to have fewer "degrees of freedom" in some sense at its disposal. However, given that the state of a recurrent network is encoded as a real-valued vector of activation levels, even for small networks there are infinitely many states to choose from. Our emphasis is therefore on studying the internal dynamics of networks attempting to solve the time delay problem to identify the limiting factors. We show that for simple delay problems, which the network is unable to solve, the system is able to learn sufficient state representations, but appears to be unable to create transitions that allow it to access those states in the correct order (or equivalently, is unable to arrange its states to suit the transitions that it can support.)

In Section 2 we provide some background to the dynamical systems approach to studying RNNs and the DFAs they learn to execute, and introduce the terminology used in the paper. We give examples of the simple pipelining case. In Section 3 we report results for cases where the number of nodes is less than the delay required. These cases force the network to use areas of the state space not required for the simpler problems. In Section 4 we examine one case in detail to see how the network partitions its state space in its best attempt at a solution, and show why the available transitions do not allow it to solve the problem with this partion. Based on this we offer some conjectures for why the network *cannot* solve the problem. In Section 5 we provide support for some of these conjectures by examining the changes that take place when particular states are removed from the input. Section 6 concludes the paper and discusses further avenues for exploration.

2 RNNs, DFA and Dynamical Systems

One of the challenges of understanding ANNs is gaining a good understanding of what the network is really doing when it learns a particular set of parameters or weights. According to Tino et al "Most of the important qualitative behaviours of a nonlinear system can be made explicit in the state space with a state space analysis." ([6], p4). The use of tools from dyanamical systems (DS) allows such an explicit view of the state space created by the activation dynamics of the network. The term "activation dynamics", when used to refer to a recurrent network, describes the relationship between the activation state of the network at the previous time step, the current input vector and the new activation state of the network. The activation functions of the network determine the particular formula that describes this relationship. The weights are the parameters of the formula.

The dynamical systems approach has been used successfully by researchers within the investigative paradigm of dynamical systems to study the interaction between the low dimensional attractor space of sensory prediction networks and the low dimensional attractor space of higher level chunking systems [8]. It has also been successfully used to analyse the inner workings of either RNNs solving specific problems or a class of RNNs [1]. Casey [1] used tools for analysing dynamical systems to show what properties a dynamical system must have, given in terms of the properties of the individual dynamical systems (created by giving the system constant input), if it is to perform an FSM calculation. These definitions are given for noisy as well as non-noisy systems and applied to example instances of second-order networks trained using Real-Time-Recurrent-Learning (RTRL).

We have based our analysis on the work of Casey [1] and Tino et al [6] who used DS analyses to track the correspondence between recurrent networks and the deterministic finite automata (DFA) they mimicked. Each network is treated as a set of discrete time dynamical systems associated with the presentation of input symbols M to the network. Casey and Tino trained their networks to recognise a language. In order to do so, their networks had to be able to model the DFA associated with accepting states for strings which are members of the language. The signal delay problem, on the other hand, requires the network to translate a set of inputs into a set of outputs based on a deterministic rule. This has an associated Moore Machine (MM; equivalent to a DFA except that the set of accept states is replaced by a set of outputs). For networks of sufficiently low dimensionality, or number of units, the i-maps (explained in further detail below) corresponding to the dynamical systems of the network can be displayed easily. This allows for a visual analysis of the state space of the network and hence the model of the DFA or MM learned by it.

The following subsections describe the signal delay problem and a typical pipeline solution in terms of the concepts described above. The relationship between the associated MM and the typical pipeline solution are demonstrated.

2.1 Signal delay: The problem and its corresponding MM

The signal delay problem involves a single binary input presented to the network that should be output by it after a specified number of discrete time steps. The number of states s in the MM that the network needs to model in order to solve the problem is directly related to the number of bits that the network needs to remember to produce the correct output, and hence to the time delay Δt (1). The activation function for an RTRL based network is given in equation (2), where $A_a(t)$ is the activation of unit a at time t, u is the number of units in the network, w_{ai} is the weight from unit i to unit a and f is a sigmoidal squashing function. One implication of this function is that the network has an inbuilt time-step delay Δt of one and therefore the minimum delay it is capable of learning is $\Delta t = 1$. Figure 1 depicts the MM associated with the signal delay problem when $\Delta t = 2$.

$$s = 2^{\Delta t} \tag{1}$$

$$A_a(t) = f(\sum_{i=1}^{u} w_{ai} A(t-1)) \tag{2}$$

Each network has a bias input that is always set to one, a binary input and u fully connected recurrent units. Each input is connected to all units within the network. One unit is selected as the output unit before training. The network is trained to perform the task of having the output mimic the input after a specified time delay. The network is considered to have successfully solved the task if the average output error is less than 0.05. We consider the network to have correctly classified a string of inputs if each output is within 0.5 of the expected output. This metric is not used in training the network however. It is used here only as a means of evaluating network solutions after training. Figure 2 depicts a 2-unit version of this network structure.

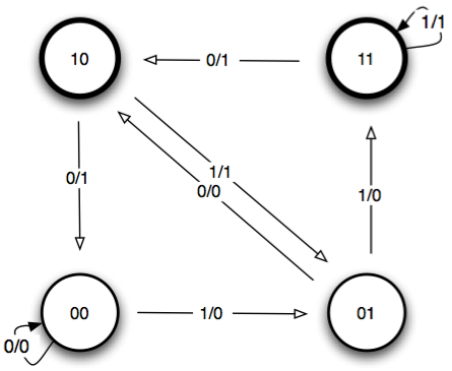


Figure 1: The MM associated with the signal delay problem for $\Delta t = 2$. The Δt binary digits labelling each state correspond to the inputs presented over the last Δt time steps (with the rightmost digit presented last). The arrows, labelled with the corresponding input and output, indicate the state transitions.

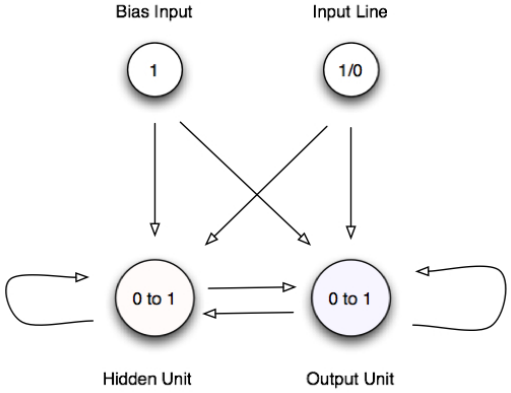


Figure 2: A 2-unit fully connected RNN. The possible activation value(s) are displayed within each input or unit. The arrowed lines indicate the weights and the direction of the connection between the units to which the weight is applied. The weights are initialised as a uniform random double between -1.0 and 1.0

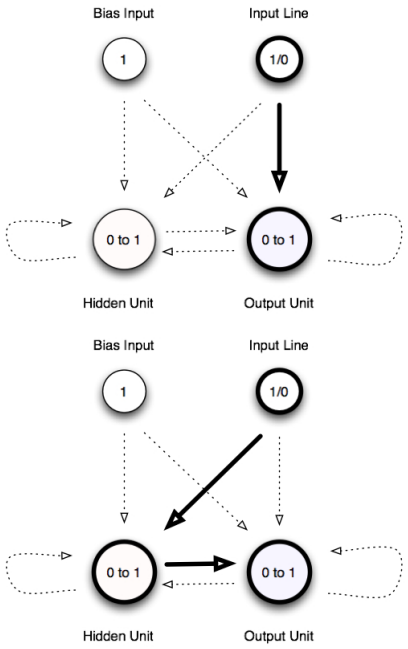


Figure 3: A representation of the structure of a 2 unit networks used in our experiments. The circles represent input lines and units, as labelled. The arrows in between indicate directional weights between the units. The boldness of the arrows corresponds the the strength of the weights. The bold lines on both the circles and arrows indicate the path an input takes over Δt steps before it is "forgotten" by the network. The first network (top) was trained to delay a binary signal by one step. The second network (bottom) was trained to delay it by two steps.

2.2 The typical "pipeline" solution: correspondence between the weight based and state-space based representations

If the network is allowed to have as many units as the number of required delay steps ($u \geq \Delta t$) then it has at least one unit available for each bit of stored information. If the units in the network are allowed only binary activation then this supports $2^{\Delta t}$ possible different states, which is equal to the number of states in the minimal MM corresponding to a time step delay of Δt (1). Intuitively this implies that a network with $u \geq \Delta t$ will be capable of modelling the states required. Figure 3 demonstrates a typical "pipeline" solution found by a two-unit network to the problem with a one and two-step time delay.

Previous analysis of the required complexity of binary units modelling DFAs has demonstrated that in the worst case the lower bounds of the number of units required to solve a DFA with s states is $\Omega(\sqrt{s})$ [7]. In our case this would indicate $\Omega(\sqrt{2^{\Delta t}})$ units. We have found, however, that with our solve criterion of error < 0.05 then for $1 \leq \Delta t \leq 10$, when the network is allowed to have at least as many units as the length of time it is required to delay the signal ($u \geq \Delta t$) it usually creates such a pipeline. This only requires $\Omega(\Delta t)$ units. This could imply that the signal delay problem is easier to model than the worst case scenario. When $\Delta t > 10$ we start to experience computational problems with obtaining results. However this could be due to the vanishing error gradients mentioned by Gers et al [4].

The dynamical systems analysis tools used by Casey [1] provide a representation of the transitions between states that occur within the network. Following Casey's definition, an $i-map$, viewed as a dynamical system, is the mapping from the $u\text{-}cube(I^u)$ into itself given a fixed input vector i and an $u\text{-}unit$ RNN. The dimensions of the cube are equal to the number of units in the network and the range in each dimension is determined by the range of activation values those units are allowed to take. The activation function provides the mapping from one point in the cube to the next The weights of the RNN are parameters of the system. An $I\text{-}map$ is the composition of the i_j $maps$ given a string of symbols $I = i_1, i_2, ...i_n$.

In order for an RNN, as a set of dynamical systems, to model the states and state transitions in a MM it has to be able to create mappings or transitions between activations such that the activation of the RNN always corresponds to the correct output. To perform robust calculations it has to be able to stay within some specified level of error ϵ of the required activation at time $t + 1$ if it is within some specified level of error ϵ of the current state at time t. To do this it has to create classes of transitions that behave the same way in terms of moving from some continuous area to another continuous area. What we mean by continuous is that within the area there are no transitions that do not behave the same way as the surrounding transitions. When we refer to a

state within an RNN, as opposed to a particular location in the state space, we are referring to a range of locations in the state space, such that the transitions from any point within that range will be of the same class as any other point in that range for any input string. In other words for those points in the state space the RNNs dynamical systems (created by holding each different possible I constant) will have the same class of transitions. The description of these concepts is based on the formal definition provided by Casey [1].

An important feature of RNN states is that they can either have the property of being transient states (known as transients) or attracting states (known as attractors) of any of the I-maps corresponding to the RNN. Casey [1] provides a precise mathematical definition of such attractor and transient states. A basic summary of this definition is that an attractor of an I-map is a state having the property that once the activation of the dynamical system is within that state it will not leave it (that is it will not transition to another state) provided the underlying dynamical system continues to receive the input string I. A transient, on the other hand, is a state having the property that once the dynamical system is in that state the application of the input string I *will* cause the dynamical system to transition to another state.

Casey [1] notes several features of the correspondence between properties of RNN states and properties of the MMs they model. For example, cycles in a DFA will correspond to attractors in the transition maps of the dynamical system created by the RNN that models it. A cycle occurs when a MM, starting in state s_i transitions to s_i after reading in a particular input string. For a MM corresponding to a signal delay problem of delay Δt there will exist a cycle in the MM for every input string of length Δt. This has implications for some of the properties an RNN, successfully trained to solve the signal delay problem, will require. If an RNN correctly models the MM then for every state of the MM there exists a corresponding state of the RNN that is an attractor of some I-map, where I is of length Δt.

A visualisation tool, useful for small networks and referred to here as an activation map, is to plot the activations of the network in u dimensions over a number of time steps. This provides a reasonable view of the boundaries of the states modelled by the network.

Figure 4 displays the appropriate section of the MM overlaid on the vector field for the 0-map and 1-map of a solution found by one of our 2 unit networks to the delay problem with $\Delta t = 2$. The correspondence between the MM and the dynamical system defined by the network becomes very clear. The vector field display, for example, provides a visualisation of the attractors of the i-maps. For the 0-map there is an attractor located in the bottom left (corresponding to state 00) of the vector field. For the 1-map there is one located in the top right (corresponding to state 11). These attractors correspond to cycles of length one of the MM when it is in state 00 or 11. Overlaying the activation map on the i-map compliments the vector field display by providing a better understanding of the practical implications of the particular vector field. Figure 5 displays the activation states visited by the network over 5000 consecutive time steps for a solution to the same problem. If the 2 unit network solution were exactly binary (that is each unit was allowed to take on only activations 0 or 1) then the activations of the network would all be in the corners of the space.

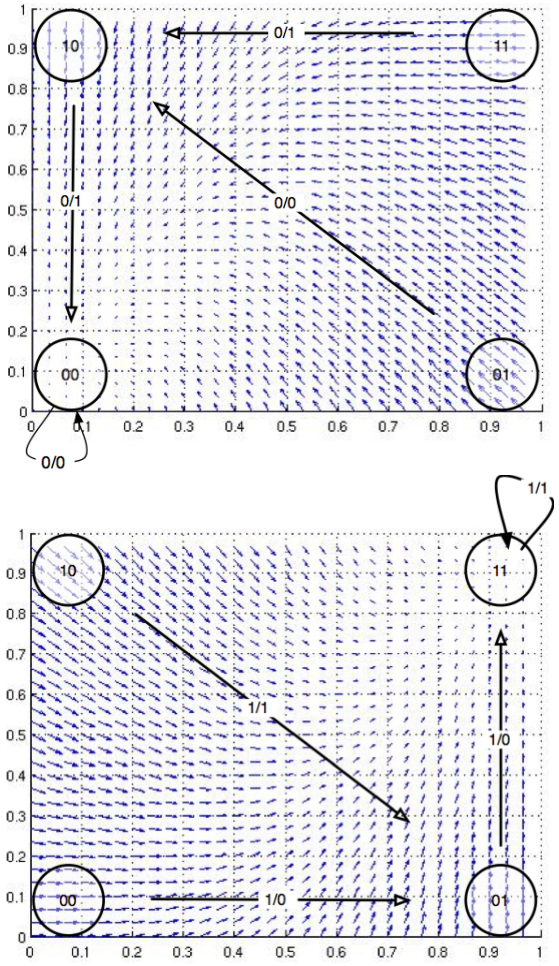


Figure 4: The vector fields of the 0-map (top) and 1-map (bottom) of a solution to the signal delay problem for $\Delta t = 2$. Notice how the vector field learned by the RNN generates the transitions corresponding to the MM. The MM transitions corresponding to an input of 0 have been overlaid on the 0-map to emphasise this relationship. Equivalently the transitions corresponding to an input of 1 have been overlaid on the 1-map.

3 Forcing the Network Away From a Pipe-Line

The recurrent networks described above can model the MM associated with the signal delay problem of delay Δt using close to binary activations for their inputs provided they have at least Δt units. However the fact that the activations of units are continuous implies that the network is capable of representing an infinite number of states. It could do this by partitioning its transitions appropriately, or in other words, by creating attractors for each possible input string I of length Δt in regions of the u-cube away from the corners. Theoretically then it could learn to represent more complex MMs than those corresponding to a pipeline solution provided it can learn the required states and their transitions with sufficient accuracy. With this in mind we designed signal delay experiments that force the network to make use of more of the state space available to it. This was initially achieved by reducing u to $\Delta t - 1$.

Table 1 displays the results of training networks of size $\Delta t - 1$ to delay a signal for Δt steps over 10 trials each. This was done for all Δt where $3 \leq \Delta t \leq 11$ (larger networks with larger time delays did not ter-

Table 1: The results of training networks on the signal delay problem where ($u = \Delta t - 1 \quad | \quad \Delta t = 1, 2, \ldots, 11$). Termination of training occurred either when the average network error was below 0.05 or the maximum number of training runs was reached. The labels expand to number of units / time delay, solve rate over 10 trials, training iterations to solve, mean minimum error reached if unsolved, time to reach mean minimum error if unsolved, mean classification accuracy (CA) when solved, and mean CA when not solved. The criterion for solving the problem was mean error < 0.05 over 100 time steps.

$u/\Delta t$	**Slv Rate**	**Slv Time**	**MER(unslvd)**	**MER Time(unslvd)**	**CA - slvd**	**CA - not slvd**
2/3	0.0	NA	0.293696	20,000,017	NA	0.780083
3/4	0.0	NA	0.173661	11,585,450	NA	0.872621
4/5	0.2	8,994,213	0.099954	7,207,508	0.978607	0.953271
5/6	0.1	7,015,836	0.094973	7,036,484	0.979563	0.949131
6/7	0.5	7,478,422	0.081307	7,461,881	0.974914	0.961935
7/8	0.8	6,751,568	0.099986	5,051,617	0.978364	0.955215
8/9	1.0	7,386,683	NA	NA	0.982654	NA
9/10	1.0	8,732,683	NA	NA	0.994315	NA
10/11	0.4	11,068,611	0.249964	10,723,936	0.998651	0.813540

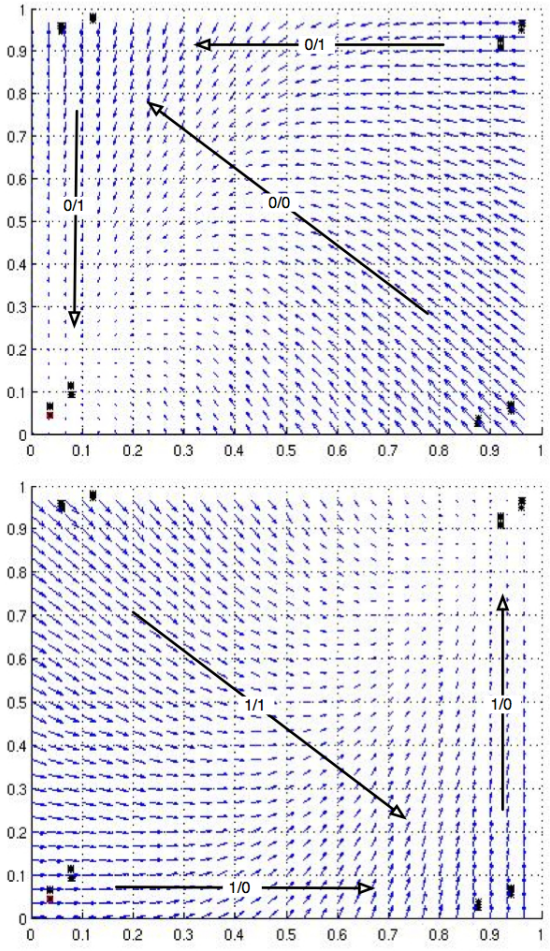


Figure 5: The vector fields of the $0 - map$ (top) and $1 - map$ (bottom) of a solution to the signal delay problem for $\Delta t = 2$. The activation values of the network (dark patches) over 5000 consecutive time-steps are plotted over the top. All but one of the activation values are within 0.2 of the correct output. The incorrect output occurs only because the network was not initialised to a correct state. The network manages to correct this error very quickly.

minate appropriately due to computational limitations). The general trend depicted in the table is that larger networks are better able to accommodate the extra bit of information they are required to remember than smaller networks. The smaller networks, where $u < 4$, do not solve the problem on any trials. The networks always learn the problem at least partially, producing better than chance levels of correct classification, and the overall accuracy of the trained networks, in general, improves as u increases even for those cases where the network does not reach the error criterion where it is considered to have solved the problem.

Intuitively it seems as though the larger networks have more "degrees of freedom" available to accommodate the extra states and/or state transitions required. However even if this is the case, and it is not obvious where these degrees of freedom would come from, we know that the u-cube corresponding to our RNNs can be partitioned infinitely. We don't know, at this stage, that it can learn or even model appropriate partitions of transitions that correspond to the required MM. Section 4 contains a closer examination of the particular case where $u = 2$ and $\Delta t = 3$ referred to as the 2-3 condition with the aim of elucidating the cause of the networks' ability to consistently solve the 3 step delay problem only partially.

4 The 2-3 Condition: What the network has to learn and what it actually learns

In order for a recurrent network to solve the 2-3 problem it has to be able to mimic or model the MM associated with the signal delay problem. Figure 6 depicts this MM. It is worth noting several features of the MM in terms of the problems they pose the network. First there are eight states in the MM that will have to be represented. Secondly, to achieve the correct output classification, four of these states (depicted in Figure 6 in bold) must be located in the top half of the activation space of the output unit whilst the other four must be located in the bottom half (denoting a 0 or a 1 output respectively). As the mean-squared-error is used to train the network it will attempt to position states as close to the appropriate extreme (1 or 0) as it can. Finally the network must be able to make all the necessary transitions.

Figure 7 displays the $0 - map$ and $1 - map$ of a solution to the 2-3 problem. The maps are overlaid with the activation state visited by the network over 5000 successive time steps. The lighter areas correspond to the times the network output had the correct classification ($error < 0.5$) and the darker areas those where it did not. All 2-3 condition trials produced almost identical $i - maps$.

The activation map appears to be divided into an appropriate number of states. This suggests that the

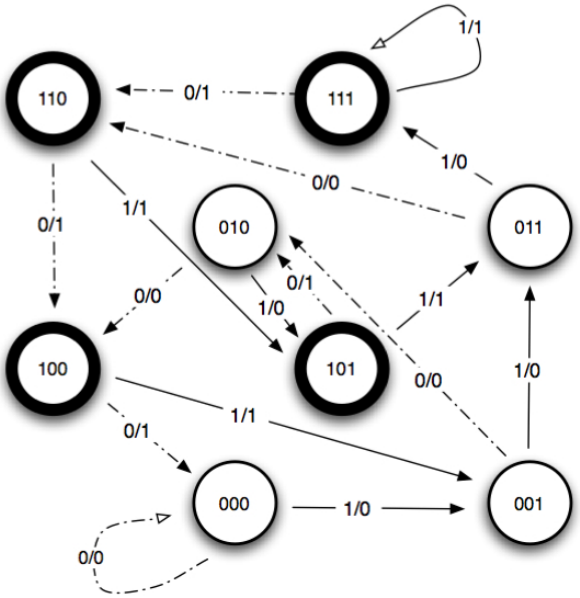


Figure 6: The MM for the signal delay problem where $\Delta t = 3$. The states depicted in bold are those where the network is expected to output a one on the next transition. The others output a zero

network managed to partition its transitions into an appropriate number of classes. We confirmed that these sections correspond to each of the states within the MM (marked on the maps) and that the network transitions between these states in the appropriate order.

While the network seems able to partition the activation space into appropriate state, as we have seen in Figure 7 it is still unable to solve the task. The problem seems to be that the network cannot arrange the transition partitions so that the states are also positioned in such a way that correct classification is achieved. The networks do not even arrange the states so that there is no overlap, along the output axis or dimension, between states where the output should be 0 at the next transition and those where it should be 1 at the next transition. Closer examination of the network outputs when long strings are processed shows that there is some overlap between some of the states, but that after a few time steps the network *recovers* from it's lapse. Thus the network does not robustly model all the states.

If the problem for the network is arranging the appropriate relationship between state transition partitions rather than representing the required number of partitions then it would be useful to have some idea of why these transitions or transition relationships are so difficult for the network. One possibility is that there are simply too many transitions for the network to arrange accurately or robustly. The fact that previous analyses of the computational complexity of various types of RNNs have yielded lower bounds in terms of the number of states and/or the length of the input alphabet suggests that this is usually an important factor. Another possibility is that there is some quality of the particular transitions involved in this problem that make things difficult, potentially impossible, for the 2 unit network to solve.

In his analysis of what properties a network has to achieve in order to model a particular FSM Casey [1] defines two classes of computation, *transient* and *hysteretic*, that may be required to model different FSMs. Definitions for these terms are given in 4.1 and 4.2. Casey comments that the number of states is unlikely to be a good measure of complexity and proposes that other measures such as the number of behaviours (increased with greater levels of hysteretic computation) are likely to be more important factors. He provides examples where hysteretic and transient computation add to the complexity of the problem.

Definition 4.1 *Transient computation is computation that must be performed using transients (or transient states) of the I-map. A transient of the I-map is a point that is not in the set $R_\epsilon(f)$, which is the set of all points such that if you start at that point (call it x), given some number of applications of the input string I to the map function f the dynamical system will have returned to within ϵ of x.*

Definition 4.2 *Hysteretic computation is computation that can be performed using periodic orbits (of periods ≥ 1) of the I-map of an RNN. Casey defines a periodic orbit of period n of map I as an n-cycle of the I-map meaning that when I is read in n times the RNN will be in its starting state. An I-map is hysteretic if it has more than one ϵ-pseudoattractor. For our purposes this is an attracting state or attractor of the I-map where once activation enters this area it does not leave it given the continued application of the input string I. See [1] for a more exact definition. An RNN is hysteretic if for some string of inputs I, its I-map is hysteretic.*

In his examples, Casey noted that for one particular problem it was computation that was transient for the 0 and 1 maps that caused the increased difficulty in solving the problem. He did this by proving that for a particular problem (recognising the language of strings beginning with 0 and having no 00 or 11 substrings) it was the transient computation that limited the conditions under which a one-unit network could learn an appropriate model. A comparison of two languages whose FSMs had the same number of states and transitions but different numbers of I-maps with hysteretic computation found that the problem with higher levels of hysteretic computation resulted in a lower solve rate and longer training times.

An examination of the typical solution to the 2-3 problem demonstrates that when our networks fail it always tends to be on the same set of state transitions. In particular it seems to fail most often on transitions involving the 010, 101, 100 and 011 states. The consistency of this finding seems to suggest that some transitions are indeed more difficult than others in this case. If we classify the transitions in the MM that correspond to the 3 step signal delay problem we see that none of the I maps of a minimal solution to the problem require more than one attractor and so the problem does not require hysteretic computation. However for the 0 and 1 maps there are seven transient states. Only when the network is in state 000 will 0 input mean that the network should not transition to a new state but should stay in the same state. The same is true of a 1 input when the network is in state 111. Thus removing a state corresponding to 000 or 111 should result in less improvement for the network than removing one of the other states if transient computation in the 0 or 1 map is more difficult than non-transient computation.

Another possibility is that the longer a transition has to be, in terms of the Euclidean distance between the starting and ending states of a transition, is important for how easy it is for the network to model that transition accurately. For our particular problem, the only limitation placed on the network in terms of how long transitions have to be is that during

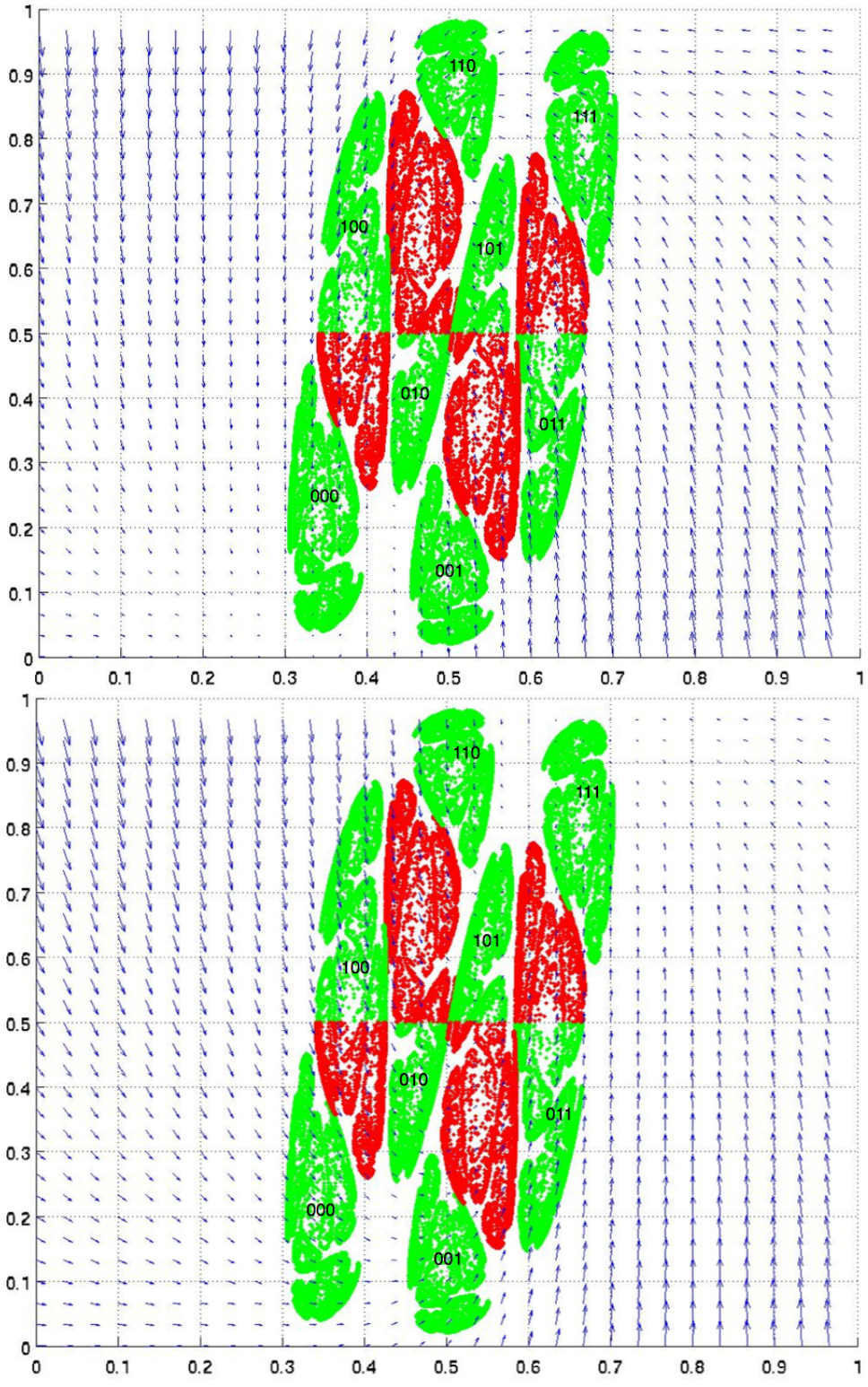


Figure 7: The vector fields of the 0-map (top) and 1-map (bottom) of an attempt at the 2-3 problem. The activation map produced by running the network continuously for 5000 time-steps is overlaid. To create this map the activation of the network at each time step is plotted using a point, whose location is determined by the hidden unit along the x-axis and the output unit along the y-axis. Points corresponding to activations that were within 0.5 of the expected output are plotted in green and those corresponding to activations that were 0.5 or more away from the expected output (0 or 1) are plotted in red.

training it is "rewarded" for placing states as close to the required output, along the output dimension, as possible. As the output is required to be either 0 or 1, the network is encouraged to place states where the output at the next transition should be 0 as far away as possible along the output dimension from states where the output at the next transition should be 1. As an explanation, this theory seems to fit well with the observation that the transitions the network seems to fail on most often are those that involve a transition between states in different output regions.

One observable feature of the i-maps in figure 7 that may be relevant to the network's apparent difficulty in learning to model the transitions involving states 010, 101 100, and 011, is that to correctly transition between these states, the network seems to be required to transition in a different rotational direction from the one used for the majority of the transitions between states. The network has arranged it's transitions so that activation moves in an anticlockwise lopsided ellipse for both the 0 and 1 maps. The difference in transitions arising from the application of a 0 or 1 input seems to be created by having slightly different axes of rotation for each of the maps. The transitions between 101 and 010 look like they might conceivably be performed using anticlockwise transitions with an adjusted map. However, the sequence 010, 101, 011 for the 1-map requires clockwise rotation. Similarly the transition sequence 101, 010 and 100 for the 0-map also requires clockwise rotation. It is possible that the degrees of freedom available to the 2 unit network limit the extent to which the direction of transitions can vary along a given dimension. This potential contribution to the difficulty of representing transitions concerns the relationship between the transitions rather than being a property of a single transition partition. This makes it harder to say that it leads to particular transitions being more difficult than others.

Section 5 describes an experiment aimed at supporting the contention that the type of transitions is an important factor in the difficulty the network has in modelling the appropriate MM for the 3 step signal delay problem.

5 Excluding patterns

If it is only the number of transitions that makes this problem difficult for the network to solve then the removal of a single state from the input and test data (by making sure that a particular combination of sequential inputs never occurs), which necessarily results in the removal of 2 transitions, should correspond to improved performance by the network. If it is only a question of the total number of transitions then the choice of state to remove should have no effect on the average ability of the networks to solve the problem. Alternatively, if there are qualitative differences in the difficulty of modelling different types of transition then potentially this will not be the case. In order to test whether the removal of more difficult transitions and consequently states will make the problem easier for the network than the removal of less difficult states we had to be able to classify states as being "easy" or "hard" based on whether their corresponding transitions were considered to be easy or hard. We used the first two explanations for the network's difficulty in solving the 2-3 problem, provided above, to make this classification. States are classified as easy or hard only if they would be considered easy or hard for both the first and second explanations. This resulted in the set of easy states being 000 and 111 as these states are the only states that are transients of only one of the 0 and 1 maps rather than both. Also the transitions leading from these states do not involve a change in the required output. The set of hard states includes 010, 101, 100, and 011 as these states, in addition to being transients of both the 0 and 1 maps lead to transitions that require that the output at the next step is different from the output at the current step. For the states 010 and 101 the required outputs from the state following the next state will be different once again from the required output of the previous state.

The sequential nature of the problem meant that only the states 000, 111, 010 and 101 could be removed from the possible input set without making it impossible to reach other states. Thus only the states 010 and 101 were used to represent hard states in our experiments. Each experiment was composed of 30 trials. Table 2 displays the results over 30 trials for each of the four patterns (000,111,010,101).

One of the most noticeable features of the results in Table 2 is that removing one state did not make the problem easy for the network to solve in any of the conditions. However it does seem to have improved the overall accuracy of the trained networks for both easy and hard states implying that there was some reduction in problem difficulty.

Another notable feature is that, although low, the solve rate for networks trained on the problem with a hard state removed is above zero and it is the only condition for which this is the case. The minimum error reached when networks failed to solve problems in this condition is also noticeably lower than for the condition with an easy state or no state removed. In general these results seem to provide support for the conjecture that the distinctions we made between easy and hard states were real and that the apparent difficulty of the transitions impacted the ability of the network to solve the problem.

6 Conclusion

Previous proofs of the computational limitations of neural network complexity seem to describe these limitations solely in terms of the number of states within the DFA the network is modelling and in some cases also in terms of the length of the input alphabet of the DFA [9]. However as Casey [1] and Tino et al [10] note this prognosis seems incomplete without reference to the impact of the quality of the transitions.

The analysis and experiments we described above attempt to address this issue for a specific instance of the signal delay problem. We suggested that the problem for the network lay in arranging the states spatially so that it can learn the transitions, or equally that given the spatial arrangements of states it *was* capable of learning it could not learn to make the required transitions.

Based on our study of the activation maps and i-maps of networks with time delay greater than the number of nodes, we suggested several possible explanations for why some of the transitions of the MM associated with the problem might be more difficult for the network to represent than others. In follow up experiments to test these hypotheses we found that there was, in fact, a difference between the accuracy of networks trained and tested on the problem with an easy versus a hard state removed. This is consistent with Casey's contention that some transitions are harder than others to learn.

There are several major limitations to our study that limit the conclusions that can be drawn from our findings. One such limitation is that our methodology does not provide enough information to make a real distinction between the difficulty involved in learning to solve the problem versus the difficulty of actually

Table 2: The results of training networks on the 2-3 problem with different states removed. The labels expand to; pattern removed, solve rate over 30 trials, steps to solve, mean minimum error reached, time to reach mean minimum error if unsolved, mean classification accuracy (CA) when solved, and mean CA when not solved. The criterion for solving the problem was mean error < 0.05 over 100 time steps.

Pat Ex	Slv Rate	Slv Time	MER(unslvd)	MER Time	CA - slvd	CA - not slvd
None	0.0	NA	0.293696	20,000,017	NA	0.780083
000	0.0	NA	0.204928363	16158806	NA	0.844467497
111	0.0	NA	0.127827034	20000017	NA	0.91791148
010	0.267	16889206	0.067628225	20000017	0.996369295	0.98528857
101	0.3	17393607	0.075848266	18517316	0.992392808	0.97816637

performing the computation. The consistency of the pattern of states learned in the 2-3 problem gives us some confidence that the learning is working effectively. However it does not rule out the possibility that this solution is a local minimum. The examples presented by Casey [1] also suffer from this limitation.

The other main problem is that our method of removing states did not enable us to remove all states individually. This meant that we were unable to select states for removal that would enable us to distinguish between some of the possible explanations for why some transitions would be harder than others. For example, it would have been useful to compare network performance with one of the states 110 and 001 removed to the performance with one of the states 010 and 101 removed. This would have allowed us to compare the relative contribution of transition length and the amount of transient computation.

In future, alternative methods such as combining two states may allow for removal of arbitrary states in the signal delay problem. Alternatively, the selection of a problem more amenable to the removal of arbitrary states might enable experiments that provide a better insight into the relative importance of these factors. The possibility that the number of changes in the direction of transitions along particular dimensions is important may need to be addressed with an alternative experimental structure. The need to manipulate the relationship between transitions may mean that solutions to different problems with the same number of states and input alphabet but different types of transition patterns may have to be compared.

A final and very important consideration for interpreting these results is that it is hard to know how they would scale up from small two unit networks. It is well established that understanding the behaviour of small dynamical systems often does not lead to an understanding of how a larger scale version will behave [10]. One strength of the proofs provided previously concerning the complexity relationship between the number of nodes required and the number of states in the network is that these results are scalable [7].

Future work in exploring the impact of the quality of required transitions would need to attempt to address this as well as the other issues mentioned above. However we feel that the findings presented here provide some interesting support for the idea that the quality of transitions is of interest in assessing the computational capabilities of RNNs.

References

[1] Casey, M.: The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction. Neural Computation **8** (1996) 1135–1178

[2] Elman, J.L.: Finding structure in time. Cognitive Science: A Multidisciplinary Journal **14**(2) (1990) 179–211

[3] Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation **9** (1997) 1735–1780

[4] Gers, F.A., Schmidhuber, J., Cummins, F.: Learning to forget: Continual prediction with LSTM. Neural Computation **12** (2000) 2451–2471

[5] Cummins, F.: Representation of temporal patterns in recurrent networks. In: Proceeding of the 15th Annual Conference of the Cognitive Science Society. (1994)

[6] Tino, P., Horne, B.G., Giles, C.L., Collingwood, P.C.: Finite state machines and recurrent neural networks - automata and dynamical systems approaches. Technical Report UMIACS-TR-95-1 and CS-TR-3396, Institute for Advanced Computer Studies, University of Maryland, College Pare, MD (1995)

[7] Sima, J., Orponen, P.: General-purpose computation with neural networks: A survey of complexity theoretic results. Neural Computation **15** (2003) 2727–2778

[8] Fujita, M.: Intelligence dynamics: An overview. In: International Workshop on Synergistic Intelligence Dynamics, Synergistic Intelligence Project, Sony corporation (2006)

[9] Carrasco, R.C., Oncina, J., Forcada, M.L.: Efficient encoding of finite automata in discrete-time recurrent neural networks. In: Artificial Neural Networks, 1999, ICANN 99. Ninth International Conference on. Volume 2. (1999) 673–677

[10] Tino, P., Horne, B.G., Giles, C.: Fixed points in two-neuron discrete time recurrent networks: Stability and bifurcation considerations. Technical Report UMIACS-TR-95-51 and CS-TR-3461, Institute for Advanced Computer Studies, University of Maryland, College Park, MD (1995)

Automatic Thesaurus Construction

Dongqiang Yang | David M. Powers
School of Informatics and Engineering
Flinders University of South Australia
PO Box 2100, Adelaide 5001, South Australia
Dongqiang.Yang|David.Powers@flinders.edu.au

Abstract[1]

In this paper we introduce a novel method of automating thesauri using syntactically constrained distributional similarity. With respect to syntactically conditioned co-occurrences, most popular approaches to automatic thesaurus construction simply ignore the salience of grammatical relations and effectively merge them into one united 'context'. We distinguish semantic differences of each syntactic dependency and propose to generate thesauri through word overlapping across major types of grammatical relations. The encouraging results show that our proposal can build automatic thesauri with significantly higher precision than the traditional methods.

Keywords: syntactic dependency, distribution, similarity.

1 Introduction

The usual way of automatic thesaurus construction is to extract the top *n* words in the similar word list of each seed word as its thesaurus entries, after calculating and ranking distributional similarity between the seed word and all of the other words occurring in the corpora. The attractive aspect of automatically constructing or extending lexical resources rests clearly on its time efficiency and effectiveness in contrast to the time-consuming and outdated publication of manually compiled lexicons. Its application mainly includes constructing domain-oriented thesauri for automatic keyword indexing and document classification in Information Retrieval, Question Answering, Word Sense Disambiguation, and Word Sense Induction.

As the ground of automatic thesaurus construction, distributional similarity is often calculated in the high-dimensional vector space model (VSM). With respect to the *basic elements* in VSM (Lowe, 2001), the dimensionality of word space can be syntactically conditioned (i.e. grammatical relations) or unconditioned (i.e. '*a bag of words*'). Under these two context settings, different similarity methods have been widely surveyed, for example for '*a bag of words*' (Sahlgren, 2006) and for grammatical relations (Curran, 2003; Weeds, 2003). Moreover, the framework conducted by Padó and Lapata (2007) compared the difference between the two settings. They observed that the syntactically constrained VSM outperformed the unconditioned one that exclusively counts word co-occurrences in a ±*n* window.

Given the hypothesis that similar words share similar grammatical relationships and semantic contents, the basic procedure for estimating such distributional similarity can consist of (1) pre-processing sentences in the corpora with shallow or complete parsing; (2) extracting syntactic dependencies into distinctive subsets or vector spaces (**X**s) according to head-modifier, including adjective-noun (**AN**) and adverb or the nominal head in a prepositional phrase to verb (**RV**) and grammatical roles including subject-verb (**SV**) and verb-object (**VO**); and (3) determining distributional similarity using similarity measures such as the Jaccard coefficient and the *cosine*, or probabilistic measures such as KL divergence and information radius. On the other hand, without the premise of grammatical relations in sematic regulation, calculating distributional similarity can simply work on word co-occurrences.

Instead of arguing the pros and cons of these two context representations in specific applications, we focus on how to effectively and efficiently produce automatic thesauri with syntactically conditioned co-occurrences.

Without distinguishing the latent differences of grammatical relations in dominating word meanings in context, most approaches simply chained or clumped these syntactic dependencies into one unified context representation for computing distributional similarity such as in automatic thesaurus construction (Hirschman et al., 1975; Hindle, 1990; Grefenstette, 1992; Lin, 1998; Curran, 2003), along with in Word Sense Disambiguation (Yarowsky, 1993; Lin, 1997; Resnik, 1997), word sense induction (Pantel and Lin, 2002), and finding the predominant sense (McCarthy et al., 2004). These approaches improved the distributional representation of a word through a fine-grained context that can filter out the unrelated or unnecessary words produced in the traditional way of '*a bag of words*' or the unordered context, given that the parsing errors introduced are acceptable or negligible.

It is clear that these approaches, based on observed events, often scaled each grammatical relation through its frequency statistics in computing distributional similarity, for example in the weighted (Grefenstette, 1992) or mutual information based (Lin, 1998) Jaccard coefficient.

Although they proposed to replace the unordered context with the syntactically conditioned one, they have overlooked the linguistic specificity of grammatical relations in word distribution. Except for the extraction of syntactically conditioned contexts, they in fact make no differentiation between them, which are similar to computing distributional similarity with unordered context. The advantage of using the syntactic constrained context has not yet been fully exploited when yielding statistical semantics from word distributions.

To fully harvest the advantages of computing distributional similarity in the syntactically constrained contexts, we proposed to first categorize contexts in terms of grammatical relations, and then overlapped the top n similar words yielded in each context to generate automatic thesauri. This is in contrast to averaging distributional similarity across these contexts, which is commonly adopted in the literature.

2 Context interchangeability of similar words

Word meaning can be regarded as a function of word distribution within different contexts in the form of co-occurrent frequencies, where similar words share similar contexts (Harris, 1985). Miller and Charles (1991) propose that word similarity depends on to what extent they are interchangeable across different context settings. The flexibility of one word or phrase substituting another indicates its extent to be synonymous providing that the alternation of meaning in discourse is acceptable. We calculated distributional similarity in different syntactic dependencies such as subject-predicate and predicate-object. Given the interchangeability of synonyms or near-synonyms in different contexts, semantically similar words derived with distributional similarity should span at least two types of syntactically constrained contexts. In other words, once we can derive the thesaurus items from each dependency set, the final thesaurus comprises the intersection of the items across any two types of dependency sets.

The heuristic of deriving automatic thesauri with the interchangeability of synonyms or near-synonyms in contexts (‘***any two***’) can be expressed:

- Nouns: $\bigcup_{i,j}(S_i \cap S_j)$ where *i* and *j* stand for any two types of dependency sets in terms of grammatical relations: **AN**, **SV**, and **VO**.
- Verbs: $\bigcup_{i,j}(S_i \cap S_j)$ where *i* and *j* stand for any two of **RV**, **SV**, and **VO**.

where for a given word, *S* is the thesaurus items produced through distributional similarity in a single dependency set. Note that we also used the heuristics of ‘***any three***’ and ‘***any four***’ to construct automatic thesauri, but found most target words had no distributionally similar words under these stricter conditions than ‘***any two***’. We did not attempt to demonstrate the conditions here.

We similarly hypothesized the union of all grammatical relations from the co-occurrence matrices as a baseline (‘***all***’), which compute distributional similarity with the union of all relations and can be indicated:

- Nouns: $S_{\bigcup_i}$ where *i* is one of **AN**, **SV**, and **VO**
- Verbs: $S_{\bigcup_i}$ where *i* is one of **RV**, **SV**, and **VO**

3 Syntactically constrained distributional similarity

To automate thesauri, we first employed an English syntactic parser based on Link Grammar to construct a syntactically constrained VSM. The word space consists of four major syntactic dependency sets that are widely adopted in the current research on distributional similarity. Following the reduction of dimensionality on the dependency sets, we created the latent semantic representation of words through which distributional similarity can be measured so that thesaurus items can be retrieved.

3.1 Syntactic dependency

The syntactically conditioned representation mainly rely on the following grounds: (1) the meaning of a noun depends on its modifiers such as adjectives, nouns, and the nominal head in a prepositional phrase as well as the grammatical role of a noun in a sentence as a subject or object (Hirschman et al., 1975; Hindle, 1990); and (2) the meaning of a verb depends on its direct object, subject, or modifier such as the head of a prepositional phrase (Hirschman et al., 1975). These results are partly consistent with the findings in studying word association and the psychological reality of the paradigmatic relationships of WordNet (Fellbaum, 1998).

With the hypothesis of ‘one sense per collocation’ in WSD, Yarowsky (1993) observed that the direct object of a verb played a more dominant role than its subject, whereas a noun acquired more credits for disambiguation from its nominal or adjective modifiers. As an application of the distributional features of words, Resnik (1997) and Lin (1997) employed the selectional restraints in subject-verb, verb-object, head-modifier and the like to conduct sense disambiguation.

The syntactic dependencies can provide a clue for tracking down the meaning of a word in context. Cruse (1986) points out that the semantic requirements are of two directions in head-modifier and head-complement, namely, determination (selector and selectee) and dependency (dependee and depender). The determination requirement emphasizes the dominant role of the selector in the semantic traits of a construction, while the dependency supplements some additional traits to formulate the integrity of the construction.

3.2 Categorizing syntactic dependencies

Suppose that a tuple $<w_i, r, w_j>$ describes the words: w_i and w_j, and their bi-directional dependency relation *r*. For example, if w_i modifies w_j through *r*, all such w_j with *r* to w_i form a context profile for w_i, likewise w_i for w_j. In the hierarchy of syntactic dependencies (Carroll et al., 1998), the major types of grammatical relationships (*r*) can be generally clustered into:

- **RV**: verbs with all verb-modifying adverbs and the head nouns in the prepositional phrases;
- **AN**: nouns with noun-modifiers including adjective use and pre/post-modification;
- **SV**: grammatical subjects and their predicates;
- **VO**: predicates and their objects.

To capture these dependencies we employ a widely used and freely available parser[2] based on Link Grammar (Sleator and Temperley, 1991). In Link Grammar each word is equipped with 'left-pointing' and/or 'right-pointing' connectors. Based on the crafted rules of the connectors in validating word usages, a link between two words can be formed in reflecting a dependency relation. Apart from these word rules, 'crossing-links' and 'connectivity' are the two global rules working on interlinks, which respectively restrict a link from starting or ending in the middle of pre-existed links and force all the words of a sentence to be traced along links. There are in total 107 major link types in the Link Grammar parser (ver. 4.1), whereas there are also various sub-link types that specify special cases of dependencies. Using this parser, we extracted and classified the following link types into the four main types of dependencies:

- **RV**

1. *E*: verbs and their adverb pre-modifiers
2. *EE*: adverbs and their adverb pre-modifiers
3. *MV*: verbs and their post-modifiers such as adverbs, prepositional phrase

- **AN**

1. *A*: nouns and their adjective pre-modifiers
2. *AN*: nouns and their noun pre-modifiers
3. *GN*: proper nouns and their common nouns
4. *M*: nouns and their various post-modifiers such as prepositional phrases, adjectives, and participles

- **SV**

1. *S*: subject-nouns/gerunds and their finite verbs. There are also some sub-link types under S, for example, Ss*g stands for gerunds and their predicates, and Sp plural nouns and their plural verbs
2. *SI*: the inversion of subjects and their verbs in questions

- **VO**

1. *O*: verbs and their direct or indirect objects
2. *OD*: verbs and their distance-complement
3. *OT*: verbs and their time objects
4. *P*: verbs and their complements such as adjectives and passive participles

Note that except for **RV**, we define the **AN**, **SV**, and **VO** dependencies almost identically to shallow parsers (Grefenstette, 1992; Curran, 2003), or a full parser of MINIPAR (Lin, 1998) but we retrieve them instead through the Link Grammar parser.

Consider, for example, a short sentence from British National Corpus (BNC):

> *'Home care Coordinator, Margaret Gillies, currently has a team of 20 volunteers from a variety of churches providing practical help to a number of clients already referred.'*

The parse of this sentence with the lowest cost in the link grammar parser is shown in Figure 1, where LEFT-WALL indicates the start of the sentence

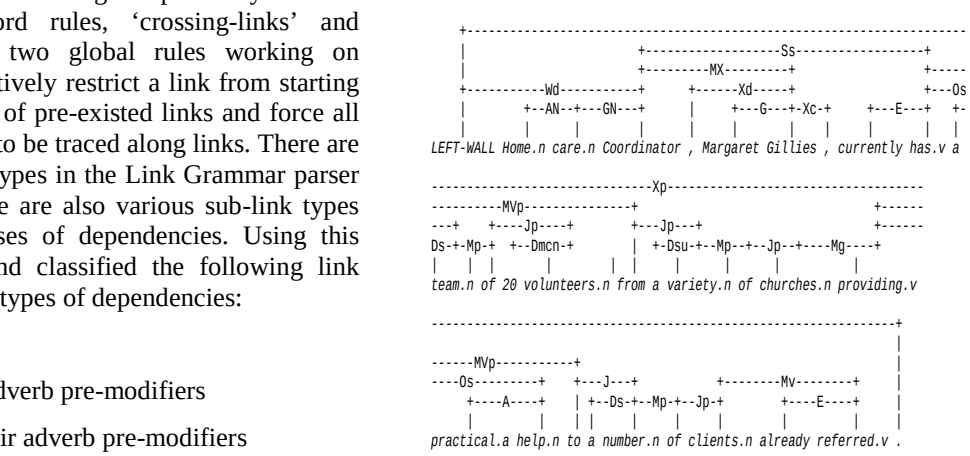


Figure 1: A complete linkage of parsing a sentence using Link Grammar

The parse of this sentence with the lowest cost in the link grammar parser is shown in Figure 1, where LEFT-WALL indicates the start of the sentence. We can classify four types of grammatical relations from this parse, namely:

- **RV**: <*currently, E, has*>, <*already, E, referred*>
- **AN**: <*home, AN, care*>, <*care, GN, coordinator*>, <*volunteer, Mp, team*>, <*church, Mp, variety*>, <*practical, A, help*>, <*client, Mp, number*>, <*referred, Mv, clients*>
- **SV**: <*coordinator, Ss, has*>
- **VO**: <*has, Os, team*>, <*providing, Os, help*>

After parsing the 100 million-word BNC and filtering out non-content words and morphology analysis, we separately extracted the relationships to construct four parallel matrixes or co-occurrence sets, denoted as R_X: $\mathbf{RV}_X$, $\mathbf{AN}_X$, $\mathbf{SV}_X$, and $\mathbf{VO}_X$ in terms of the four types of syntactic dependencies above. The row vectors of R_X denoted respectively $\mathbf{Rv}_X$, $\mathbf{An}_X$, $\mathbf{Sv}_X$, and $\mathbf{Vo}_X$ for the four dependencies. Similarly, the column vectors of R_X are denoted as $\mathbf{rV}_X$, $\mathbf{aN}_X$, $\mathbf{sV}_X$, and $\mathbf{vO}_X$ respectively.

Consider $\mathbf{SV}_X$ a *m* by *n* matrix representing subject-verb dependencies between *m* subjects and *n* verbs. We illustrate the **SV** relation using the rows ($\mathbf{Sv}_X$ or $\{X_{i,*}\}$) of $\mathbf{SV}_X$ corresponding to nouns conditioned as subjects of verbs in sentences, and the columns ($\mathbf{sV}_X$ or $\{X_{*,j}\}$) to

[2]http://www.link.cs.cmu.edu/link/

verbs conditioned by nouns as subjects. The cell $X_{i,j}$ shows the frequency of the *i*th subject with the *j*th verb. The *i*th row $X_{i,*}$ of $\mathbf{SV}_X$ is a profile of the *i*th subject in terms of its all verbs and the *j*th column $X_{*,j}$ of $\mathbf{SV}_X$ profiles the *j*th verb versus its subjects.

The parsing results are shown in Table 1, where *Dim* refer to the size of each matrix in the form of rows by columns, and *Freq* segmentations are the classification of frequency distribution, and Token/Type stands for the statistical frequencies of specific relationships with their corresponding dependency category R.

Dim \ *Freq*			1	2-10	11-20	21-30	>31
$\mathbf{AN}_X$	48.5 by	Token	1,813.7	6,243.4	1,483.1	799.8	3,617.8
	37.6	Type	1,813.7	2,040.0	103.6	32.2	44.9
$\mathbf{RV}_X$	37.4 by	Token	863.1	2,276.4	481.4	234.9	692.2
	14.2	Type	863.1	751.9	33.8	9.5	10.9
$\mathbf{SV}_X$	32.7 by	Token	511.8	1,699.4	297.8	133.3	380.7
	11.3	Type	511.8	587.4	21.0	5.4	6.0
$\mathbf{VO}_X$	6.1 by	Token	488.5	1,811.5	475.4	266.2	1,286.9
	33.3	Type	488.5	575.1	33.1	10.7	15.6

Table 1: The statistics of the syntactically conditioned matrices derived from parsing BNC (thousand)

Given different methodologies to implementing parsing, it is hardly fair to appraise a syntactic parser. Molla and Hutchinson (2003) compared the Link Grammar parser and the Conexor Functional Dependency Grammar (CFDG) parser with respect to intrinsic and extrinsic evaluations. In the intrinsic evaluation the performance of the two parsers was compared and measured in terms of the precision and recall of extracting four types of dependencies, including subject-verb, verb-object, head-modifier, and head-complement. In the extrinsic evaluation a question-answering application was used to contrast the two parsers. Although the Link Grammar parser is inferior to the CFDG parser in locating the four types of dependencies, they are not significantly different when applied in question answering. Given that our main task is to investigate the function of the syntactic dependencies: **RV**, **AN**, **SV**, and **VO**, acquired with the same Link Grammar parser, in automatic thesaurus construction, it is appropriate to use the Link Grammar parser to extract these dependencies.

3.3 Dimensionality reduction in VSM

The four syntactically conditioned matrices, as shown in Table 1, are extremely sparse with nulls in over 95% of the cells. Instead of eliminating the cells with lower frequencies, we kept all co-occurrences unchanged to avoid worsening data sparseness.

Our matrices record the context with both syntactic dependencies and semantic content. These dual constraints yield rarer events than word co-occurrences in '*a bag of words*'. However, they impose more accurate or meaningful grammatical relationships between words providing the parser is reasonable accurate.

We initially substituted each cell frequency $freq(X_{i,j})$ with its information form using $log(freq(X_{i,j})+1)$ to retain sparsity ($0\rightarrow 0$) (Landauer and Dumais, 1997). It can produce 'a kind of space effect' that can lessen the gradient of the frequency-rank curve in Zipf's Law (1965), reducing the gap between rarer events and frequent ones.

Singular Value Decomposition (SVD) often acts as an effective way of reducing the dimensionality of word space in natural language processing. A reduced SVD representation can diminish both 'noise' and redundancy whilst retaining the useful information that has the maximum variance. This approach has been dubbed Latent Semantic Analysis (LSA) (Deerwester et al., 1990; Landauer and Dumais, 1997) and maps the word-by-document space into word-by-concept and document-by-concept spaces. Note that the 'noisy' data in the raw co-occurrence matrices mainly comes from the results of wrong parsing and also redundancy exists as a common problem of expressing similar concepts in synonyms.

Typically at least 200 principal components are employed in Information Retrieval to describe the SVD compressed word space. Instead of optimising the semantic space versus other algorithms (through tuning the number of principal components in applications or evaluations), we specified a fixed dimension size for the compressed semantic space, which is thus not expected to be optimal for our experiment. We established 250 as a fixed size of the compressed semantic space. Among the singular values, the first 20 components account for around 50% of the variance, and the first 250 components for over 75%.

As is usual with the SVD/LSA application, we assume that the semantic representation of words is a linear combination of eigenvectors representing their distinct subcategorizations and senses, and that relating the uncorrelated eigenvector feature sets of different words can thus score their proximity in the semantic space.

3.4 Distributional similarity

We consistently employed the cosine similarity of word vectors as used in LSA and commonly adopted in assessing distributional similarity (Salton and McGill, 1986; Schütze, 1992). The cosine of the angle θ between vectors x and y in the *n*-dimensional space is defined as:

$$\cos\theta = \frac{x \cdot y}{\|x\|\|y\|} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i^2}_{i}}$$

where the length of x and y is ||x|| and ||y||.

Note that the accuracy and coverage of automatic term clustering inevitably depend on the size and domains of the corpora employed, as well as similarity measures. Consistently using one similarity method—the *cosine*, our main task in this paper is to explore the context

interchangeability in automatic thesaurus construction, rather than to compare different similarity measures with one united syntactic structure that combines all the dependencies together. Although taking into account more similarity measures in the evaluations may solidify conclusions, this would take us beyond the scope of the work.

4 Evaluation

4.1 The 'gold standard' thesaurus

It is not a trivial task to evaluate automatic thesauri in the absence of a benchmark set. Subjective assessment on distributionally similar words seems a plausible approach to assessing the quality of term clusters. It is practically unfeasible to implement it given the size of the term clusters. A low agreement on word relatedness also exists between human subjects.

The alternative way of measuring term clusters is to contrast them with existing lexical resources. For example, Grefenstette (1993) evaluated his automatic thesaurus with a 'gold standard' dataset consisting of Roget's Thesaurus ver. 1911, Macquarie Thesaurus, and Webster's 7th dictionary. If two words were located under the same topic in Roget or Macquarie, or shared two or more terms in their definitions in the dictionary, they were counted as a successful hit for synonyms or semantic-relatedness. To improve the coverage of the 'gold standard' dataset, Curran (2003) incorporated more thesauri: Roget's Thesaurus (supplementing the free version of 1911 provided by Project Gutenberg with the modern version of Roget's Thesaurus II), Moby Thesaurus, The New Oxford Thesaurus of English, and The Macquarie Encyclopaedic Thesaurus.

The 'gold standard' datasets are not without problem due to their domain and coverage, because they are at best a snapshot of general or specific English vocabulary knowledge (Kilgarriff, 1997; Kilgarriff and Yallop, 2000). Moreover, the organization of thesauri forces different notions of being synonymous or similar, given the etymologic trend of words and different purposes of lexicographers. For example, as 1 of 1,000 topics in Roget's Thesaurus ver. 1911, there are two groups of synonyms {*teacher*, *trainer*, *instructor*, *institutor*, *master*, *tutor*, *director*, etc.} or {*professor*, *lecturer*, *reader*, etc.} under the topic of *teacher*. They express an academic concept of being in the position of supervision over somebody. In the noun taxonomy of WordNet, the synonym of *teacher* only consists of *instructor*, affiliated with the coordinate terms (sharing one common superordinate) such as *lecturer* and *reader*, or the hyponyms such as *coach* and *tutor*, or the hypernyms such as *educator* and *pedagogue*. As for *professor* and *master*, they both distance *teacher* by three links through their hypernym *educator*.

Subject to the availability of these thesauri or dictionaries, we incorporated both WordNet and Roget's Thesaurus, freely acquired, into the 'gold standard' thesaurus. WordNet only consists of paradigmatic relations and organizes a fine-grained semantic taxonomy mainly with the relationships of syn/antonym, IS-A, HAS-A, whereas Roget's Thesaurus covers both syntagmatic and paradigmatic relations and hierarchically clusters related words or phrases into each topic without explicitly annotating their relationships.

Kilgarriff and Yallop (2000) claimed that WordNet, along with the automatic thesauri generated under the hypothesis of similar words sharing similar syntactic structures, are *tighter* rather than *looser* in defining whether they are 'synonyms' or related words. This contrasts with Roget and the automatic thesauri derived through unordered word co-occurrences. Since we accounted for distributional similarity in the syntactically conditioned VSM, the reasonable way of evaluating it is to compare our automatic thesauri to WordNet. Apart from that, to perform a systematic evaluation on the relationships among distributionally similar words, we also included Roget as a supplement to the 'gold standard', as it covers words with both paradigmatic and syntagmatic relationships.

4.2 Similarity comparison

We defined two distinctive measures to compare automatic thesauri with the 'gold standard', which are Sim_{WN} for WordNet and Sim_{RT} for Roget.

4.2.1 Similarity in WordNet

Sim_{WN} is based on the taxonomic similarity method proposed by Yang and Powers (2005; 2006). Since Yang and Powers's method outperformed most popular similarity methods in terms of correlation with human similarity judgements, we employed them in the evaluation. Given two nominal or verbal concepts: $c1$ and $c2$, Sim_{WN} scores their similarity with:

$$Sim(c1, c2) = \alpha_{str} \times \alpha_t \times \beta_t^{dist-1}, dist \le \gamma$$

- α_{str}: 1 for nouns but for verbs successively falls back to α_{stm} the verb stem polysemy ignoring sense and form; or α_{der} the cognate noun hierarchy of the verb; or α_{gls} the definition of the verb.
- α_t: the path type factor to specify the weights of different link types, i.e. syn/antonym, hyper/ hyponym and holo/meronym in WordNet.
- β: the probability associated with a direct link between concepts (type t).
- *dist*: the distance between two concept nodes
- γ: the path length *dist* is limited to depth factor γ, otherwise the similarity is 0

As for multiple senses of a word, word similarity maximizes their sense or concept similarity in WordNet.

Yang and Powers (2005) compared their taxonomic similarity metric with human judgements on the 65 noun pairs, where the cut-off point 2.36 of human similarity scores for nouns on a Likert scale from 0 to 4 divides each dataset into similar ($\geq$ 2.36) and dissimilar subsets ($<$ 2.36). We found that the cut-off of 2.36 for nouns corresponds to the searching depth limit $\gamma = 4$ in Sim_{WN},

and likewise the cut-off of 2 on the 130 verb pairs (Yang and Powers, 2006) corresponds to $\gamma = 2$. Thus for the noun candidates in automatic thesauri, we set up $\gamma = 4$, to identify similar words within the distance of less than four links. If two nodes are syn/antonyms or related to each other in the taxonomy with the shortest path length of less than 4, we counted them as a successful hit. So too is the shorter distance limit $\gamma = 2$ for verb candidates.

4.2.2 Similarity in Roget's Thesaurus

Roget's Thesaurus divides its hierarchy into seven levels from the top *class* to the bottom *topic*, and stores topic-related words under 1 of 1,000 topics. Sim_{RT} counted it a hit if two words are situated under the same *topic*.

Note that the relationships among the 'gold standard' words retrieved by Sim_{RT} are anonymous. Although WordNet only organizes paradigmatic relationships, Sim_{WN} does not distinguish in what way two words are similar, for example, IS-A, HAS-A, or a mixture of them, and only collects words within a distance from zero (syn/antonyms) to four links in WordNet.

4.3 Candidate words in the 'gold standard'

		WordNet						Roget	Total
		SA	D1	D2	D3	D4	∑		
Noun	$\mathbf{aN}_X$	462	2,825	14,244	41,483	48,625	107,639	141,102	232,181
	$\mathbf{An}_X$	458	2,887	14,278	41,940	49,267	108,830	142,218	234,424
	$\mathbf{vO}_X$	439	2,619	13,027	37,433	43,620	97,138	133,733	214,727
	$\mathbf{Sv}_X$	434	2,607	12,938	37,355	43,274	96,608	131,527	212,156
∑X		469	2,979	14,967	44,185	52,054	114,779	146,435	244,245
Verb	$\mathbf{rV}_X$	1,282	24,702	58,617			84,601	81,713	144,545
	$\mathbf{Vo}_X$	1,260	24,265	57,225			82,750	79,771	141,039
	$\mathbf{sV}_X$	1,269	24,354	57,642			83,265	80,681	142,256
∑X		1,297	25,283	60,483			87,165	83,415	148,455

Table 2: The word relatedness distribution in the 'gold-standard' across each matrix

We select 100 seed nouns and 100 seed verbs with term frequencies of around 10,000 times in BNC. The average frequency of these nouns is about 8,988.9, and 10,364.4 for these verbs. High frequency words are likely to be generic or general terms and the less frequent words may not happen in the semantic sets. The average frequency of the nouns in $\mathbf{An}_X$, $\mathbf{aN}_X$, $\mathbf{Sv}_X$, and $\mathbf{vO}_X$ is in fact decreased to 3,361.1, 5,629.1, 1,156.7, and 1,692.1, and the verbs in $\mathbf{rV}_X$, $\mathbf{Vo}_X$, and $\mathbf{sV}_X$ are decreased to 3,014.3, 3,328.9, and 1,971.8, as we only extracted syntactic dependencies from BNC. Overall, the average frequency of the nouns is about 2,959.7 across $\mathbf{An}_X$, $\mathbf{aN}_X$, $\mathbf{Sv}_X$, and $\mathbf{vO}_X$, and 3,960.9 for the verbs across $\mathbf{rV}_X$, $\mathbf{Vo}_X$, and $\mathbf{sV}_X$.

We first used Sim_{WN} and Sim_{RT} to compare each seed word to all other words from the dependency sets, namely $\mathbf{An}_X$, $\mathbf{aN}_X$, $\mathbf{Sv}_X$, and $\mathbf{vO}_X$ for nouns and $\mathbf{rV}_X$, $\mathbf{Vo}_X$, and $\mathbf{sV}_X$ for verbs, to retrieve its candidate words in the 'gold standard'. Instead of a normal thesaurus with a full coverage of PoS tags, we only compiled the synonyms of nouns and verbs that account for the major part of published thesauri and are more informative than other PoS tags. The word distribution within different distances to the 100 nouns and 100 verbs in the 'gold-standard' are listed in Table 2, where ∑X indicates the overall nouns from $\mathbf{An}_X$, $\mathbf{aN}_X$, $\mathbf{Sv}_X$, and $\mathbf{vO}_X$ and verbs from $\mathbf{rV}_X$, $\mathbf{Vo}_X$, and $\mathbf{sV}_X$ in the 'gold-standard'. For the 'gold-standard' words from WordNet, SA denotes syn/antonyms of the targets, and DI the words with exactly I link distance to targets (for nouns $I \leq \gamma = 4$; for verbs $I \leq \gamma = 2$); ∑ denotes the total number of 'gold-standard' words in each matrix; and Total means the overall number of 'gold-standard' words from both WordNet and Roget. In Table 2 the average number of 'gold-standard' words across each matrix is evenly distributed.

The agreement between the WordNet-style and Roget-style words in the 'gold-standard' across these matrices, that is, the ratio of the number of words retrieved by Sim_{WN} and Sim_{RT} in both WordNet and Roget against the total number of 'gold-standard' words, is on average 7.3% on nouns and less than 15.2% on verbs. We aggregated all the 'gold-standard' words across $\mathbf{An}_X$, $\mathbf{aN}_X$, $\mathbf{Sv}_X$, and $\mathbf{vO}_X$ for nouns, as well as $\mathbf{rV}_X$, $\mathbf{Vo}_X$, and $\mathbf{sV}_X$ for verbs, which results in 244,245 nouns and 148,455 verbs overall in the 'gold standard'. The agreement between WordNet and Roget candidates on nouns and verbs is respectively about 6.9% and 14.9%, that is to say, about 14.8% and 11.6% nouns in WordNet and Roget are of same, so are 25.4% and 26.5% for verbs. Each target noun on average owns about 1,148 WordNet, 1,464 Roget, and 2442 Total words in the 'gold standard', and each target verb 872, 834, and 1485 words respectively.

4.4 A walk-through example

For each seed word, after computing the *cosine* similarity of the seed with all other words in each dependency matrix, we produced and ranked the top *n* words as candidates. We then applied the two heuristics: '***any two***' and '***all***' on these candidates to forming automatic thesauri.

In Table 3 we exemplify the top 20 similar words of *sentence* and *attack* yielded in each dependency set and the two heuristics. Consider the distributionally similar words of *sentence* and *attack* in $\mathbf{aN}_X$ and $\mathbf{rV}_X$ for example. The words related to the linguistic sense of *sentence* consists of *syllable*, *words*, *adjective*, etc, in $\mathbf{aN}_X$, while the words with the judicial sense make up around half of the 20 words including *imprisonment*, *penalty*, and the like. The words such as *rape* and *slaughter* from $\mathbf{rV}_X$ are from the literal sense of *attack*, together with its metaphorical sense among other words like *badmouth*, *flame*, and so on.

The heuristic of '***any two***' collected the intersection of thesaurus items across these dependency sets. For example, *punishment* and *words* are the similar words to *sentence*, which respectively occurred in $\mathbf{aN}_X$ and $\mathbf{vO}_X$ as well as in $\mathbf{aN}_X$ and $\mathbf{An}_X$; *criticise* and *bomb* are the similar words to *attack*, which respectively occurred in $\mathbf{Vo}_X$ and $\mathbf{rV}_X$ as well as in $\mathbf{Vo}_X$ and $\mathbf{sV}_X$.

	Similar words
aN$_X$	*imprisonment term utterance penalty excommunication syllable words punishment prison prisoner phrase detention hospitalisation fisticuffs banishment verdict Minnesota meaning adjective warder*
An$_X$	*words syllable utterance clause nictation word swarthiness paragraph text homograph discourse imprisonment nonce phrase hexagram adjective verb niacin savarin micheas*
vO$_X$	*soubise cybele sextet cristal raper stint concatenation kohlrabi tostada apprenticeship ban contrivance Guadalcanal necropolis misanthropy roulade gasworks curacy jejunum punishment*
Sv$_X$	*ratel occurrence cragsman jingoism shiism Oklahoma genuineness unimportance language gathering letting grimm chaucer accent taxation ultimatum arrogance test verticality habituation*
any two	*imprisonment words utterance word term punishment paragraph text phrase jail verb meaning noun poem language passage sequence syllable lexicon fine*
all	*Imprisonment utterance penalty excommunication punishment prison prisoner detention hospitalisation banishment Minnesota meaning contrariety phoneme consonant counterintelligence starvation fine cathedra lifespan*

(a) The similar words to *sentence* (as a noun)

	Similar words
rV$_X$	*assault rape criticize arm slaughter abduct mortar accuse defend fire avow lash badmouth blaspheme slit singe flame kidnap persecute*
Vo$_X$	*Raid criticise bomb realign outwit beleaguer guard raze bombard criticize resemble spy pulse misspend reformulate alkalinise metastasise placard ruck glory*
sV$_X$	*ambush invade fraternize palpitate patrol wound pillage bomb billet shell fire liberate kidnap raid garrison accuse assault arrest slaughter outnumber*
any two	*assault criticize bomb ambush accuse raid fire rape bombard kidnap infiltrate patrol defend storm invade arrest garrison torture stab shoot*
all	*raid bomb assault criticize ambush accuse fire guard bombard patrol rape storm infiltrate wound kidnap criticise garrison alkalinize torture spy*

(b) The similar words to *attack* (as a verb)

Table 3: A sample of thesaurus items

4.5 Performance evaluation

Instead of simply matching with the 'gold standard' thesauri, Lin (1998) proposed to compare his automatic thesaurus with WordNet and Roget on their structures, taking into account the similarity scores and orders of similar words respectively produced from distributional similarity and taxonomic similarity. This approach can account for thesaurus resemblance under the hierarchy of WordNet or Roget, which is an apparent advantage over straight word matching.

Instead of calculating the varied cosine similarity between each target vector yielded from automatic thesaurus and from WordNet or Roget (Lin, 1998), we adapted the concept of Precision (*Pn*) and Recall-precision (*Rp*) from information retrieval to demonstrate much sensible values of precision and recall for a ranked list. Given the top *n* similar words *S* for a target *T* in an automatic thesaurus *Pn* is defined as $|S|/n$, where $|S|$ refers to the number of *S* that can be retrieved in the top *n* similar words of *T* in WordNet or Roget. *Rp* is conditioned on precision and is correspondingly defined as $|S|/\sum d(S)$, where in terms of words $d(S)$ denotes minimum distance between *T* and *S* if *S* can be located within the top *n* similar words of *T* in WordNet or Roget. Analogously for the ranked word list from an automatic thesaurus, the top *n* similar words with respect to each sense of *T* in WordNet are produced in the order of hyper/hyponyms and holo/meronyms with exhausting initially synonyms and then antonyms, whereas the top *n* words in Roget can be subsequently acquired within +/-*n* (preceding/succeeding) words from *T* in each of its category. Through these redefined precision and recall *Pn* can stand for the coverage of the automatic thesaurus on potentially arbitrary senses or categories of T and *Rp* can describe relatedness of the thesaurus on the actual sense or category of *T*.

5 Results

We took the top *n* similar words derived from each co-occurrence matrix for '***any two***' or '***all***', with *n* varying from 1 to 1000 in ten steps, roughly doubling each time. The results are shown in Table 4. We individually listed *Pn* and *Rp* values with respect to WordNet, Roget, and the union of WordNet and Roget (Total).

		'***all***'						'***any two***'					
		WordNe		Roget		Total		WordNe		Roget		Total	
N		*Pn*	*Rp*	*Pn*	*Rp*	*Pn*	*Rp*	*Pn*	*Rp*	*Pn*	*Rp*	*Pn*	*Rp*
1	noun	22.0	22.0	15.0	15.0	27.0	27.0	24.0	24.0	12.0	12.0	**28.0**	**28.0**
	verb	13.0	13.0	7.0	7.0	16.0	16.0	15.0	15.0	8.0	8.0	**20.0**	**20.0**
2	noun	31.0	35.2	19.0	23.7	36.0	**41.2**	34.0	34.0	20.0	20.0	**42.0**	37.5
	verb	39.0	31.7	9.5	12.0	40.0	34.2	48.5	34.4	11.0	13.3	**49.5**	**38.2**
5	noun	42.4	21.1	22.2	29.5	46.8	**27.1**	56.6	17.1	28.4	24.0	**63.2**	20.0
	verb	54.2	25.6	20.2	17.1	55.8	26.9	62.6	27.4	23.8	15.0	**64.0**	**28.7**
10	noun	43.4	11.8	19.4	18.5	47.5	**15.5**	56.6	10.4	26.9	17.1	**62.3**	11.0
	verb	53.3	19.5	18.0	17.5	54.7	19.6	62.3	21.7	20.9	15.9	**63.7**	**21.2**
20	noun	37.7	9.5	16.1	13.8	41.6	**9.8**	50.2	8.7	22.7	16.5	**56.0**	8.4
	verb	49.3	15.0	13.9	15.0	50.9	14.7	57.5	15.6	16.1	13.8	**59.0**	**15.4**
50	noun	29.0	8.0	11.2	11.2	32.3	**7.4**	41.4	7.2	16.7	9.5	**46.4**	6.8
	verb	43.8	11.9	10.0	10.9	45.4	11.3	49.5	12.2	11.4	9.9	**51.3**	**11.5**
100	noun	22.9	8.4	8.2	9.5	25.7	**7.4**	33.8	6.6	12.8	6.6	**38.4**	5.9
	verb	39.7	10.0	7.7	8.4	41.2	9.2	44.1	10.4	8.4	7.5	**45.6**	**9.8**
200	noun	18.6	6.9	5.9	7.8	20.9	**5.9**	26.6	6.2	8.9	6.2	**30.2**	5.5
	verb	36.0	9.3	5.9	6.5	37.4	**8.6**	39.6	9.3	6.4	6.2	**41.0**	8.5
500	noun	13.6	6.4	3.9	6.1	15.4	**5.5**	18.6	6.0	5.4	5.8	**21.0**	5.3
	verb	32.6	8.5	4.2	5.7	33.8	**7.7**	35.1	8.5	4.6	5.3	**36.4**	**7.7**
1000	noun	11.0	6.3	2.8	5.5	12.4	**5.4**	14.1	6.1	3.6	5.5	**16.0**	5.2
	verb	30.5	8.2	3.4	4.9	31.6	**7.3**	32.7	8.2	3.6	4.9	**33.8**	**7.3**

Table 4: The *precision* and *recall* in automatic thesauri under the heuristics of '*any two*' and '*all*' (percentage)

6 Discussion

6.1 '*any two*' vs '*all*'

It is clear that in terms of *Pn* measurement '***any two***' consistently outperformed '***all***' for both nouns and verbs in thesaurus construction. The improvement in the precision of the '***any two***' clusters over the '***all***' heuristic

was significant ($p < 0.05$, paired t test). This is achieved under the condition of comparable *Rp*. Before reaching the threshold 200, the overall *Rp* for verbs for '***any two***' almost stay higher than for '***all***', which is contrary in the case of nouns. Since then no noticeable difference can be observed. The reason behind this could be that some 'gold-standard' words derived from a matrix may never occur in the thesaurus entries from another matrix, which are neglected in '***any two***'.

We also extend this work to the words with intermediate (around 4,000) and low (around 1,000) term frequencies in BNC. For the 100 nouns and 100 verbs with the intermediate frequencies, 3,753.9 and 3,675.2 respectively, the average frequency of the nouns across $\mathbf{An}_X$, $\mathbf{aN}_X$, $\mathbf{Sv}_X$, and $\mathbf{vO}_X$ is 1,274.7, and the verbs across $\mathbf{rV}_X$, $\mathbf{Vo}_X$, and $\mathbf{sV}_X$ is 1,422.0. For the 100 nouns and 100 verbs with low frequencies: 824.1 and 864.6, the average frequency of the nouns across $\mathbf{An}_X$, $\mathbf{aN}_X$, $\mathbf{Sv}_X$, and $\mathbf{vO}_X$ is 297.0, and the verbs 342.2 across $\mathbf{rV}_X$, $\mathbf{Vo}_X$, and $\mathbf{sV}_X$. For the intermediate and low frequency words, the heuristic of '***any two***' still significantly outperformed the '***all***' in yielding automatic thesauri ($p < 0.05$) with higher precision.

As the threshold increasing from 1 to 1000 in Table 4, both the nominal and verbal parts of thesaurus using the heuristics of '***any two***' and '***all***' could corroborate a preference for relationships from WordNet rather than from Roget, since both *Pn* in WordNet contributed majority of the overall *Pn* in contrast to it in Roget. Note that from the figures shown in Table 2, we can observe that the overlap between WordNet and Roget is rather small, where only 14.8% of WordNet or 11.6% of Roget for nouns co-occur, so does 25.4% of WordNet or 26.5% of Roget for verbs. This could be caused by filtering out more Roget words present in the '***all***' or '***any two***' thesaurus. This trend keep unchanged even when more unrelated words could be introduced as the threshold approached 1000.

We can compare the entry of *sentence* and *attack* with the threshold of 20 in the '***any two***' thesaurus to their respective entries in the '***all***' thesaurus, that are listed in Table 3. The entry of *sentence* in the '***any two***' thesaurus constituted the top 20 similar words in Table 3 (a), they were all akin to *sentence* without any 'noisy' words such as *Minnesota* and *counterintelligence* in the '***all***' thesaurus. So did *attack* in Table 3 (b), which comprised near-synonyms after filtering out the unrelated words such as *alkalinise* in the '***all***' thesaurus. However, some truly related words were also missed out in the '***any two***' thesauri, for example, the similar words *penalty* and *banishment* to *sentence* in the '***all***' thesaurus, as well as *guard* and *wound* to *attack*. This can be partly complemented through increasing the threshold. Even with the threshold 50, the overall thesaurus entries of were still acceptable with approximately 50% of total precision.

6.2 The predominant sense

Word senses in WordNet are ranked by their frequencies, where the first sense often serves as the predominant sense of a word. The predominant sense often serves as a back-off in sense disambiguation. To study the sense distribution of the words in automatic thesaurus, we also calculated *Pn* on the condition of extracting the 'gold-standard' words exclusively related to the first sense of a target (*First*), in contrast to all the senses.

Overall the precision of *First* sense is not less than 50% of the precision of all sense for both nouns and verbs in the '***any two***' heuristic. This implies that distributionally similar words derived using the '***any two***' heuristic are more semantically related to the first sense of a target, around 50% or more, than other senses. Even in the '***all***' heuristic around 50% of the words that match a 'gold-standard' for any sense, hold semantic relatedness with the first senses of targets.

The unbalanced sense distribution among the thesaurus items shows the uneven usages of words with respect to the Zipf's Law (1965). Kilgarriff (2004) also noted Zipfian distribution of both word sense and words when analysing the Brown corpus and BNC. The predominant sense of a word can be formed through their distributionally similar words instead of laborious sense annotation work, which serves as an important resource in sense disambiguation.

6.3 Distributional similarity and semantic relatedness

Semantic similarity is often regarded as a special case of semantic relatedness, while the latter also contains word association. Distributional similarity consists of both semantic similarity and word association between a seed word and candidate words in its thesaurus items, except for the 'noisy' words (due to the parsing or statistical errors) that hold no plausible relationships with the seed. Consider the distributionally similar words of *sentence* produced in $\mathbf{aN}_X$ in Table 3 (a) for example. Only three words, namely *term*, *phrase*, and *verdict*, were connected with *sentence* through the similarity measurement of Sim_{WN} in WordNet, whereas 14 words such as *phrase* and *penalty* shared the same topics with *sentence* in Roget. The noun *sentence* consists of three senses in WordNet,

- *sentence#n#*1: a string of words satisfying the grammatical rules of a language
- *sentence#n#*2: (criminal law) a final judgment of guilty in a criminal case and the punishment that is imposed
- *sentence#n#*3: the period of time a prisoner is imprisoned

The word *sentence* is also located in Section 480 (*Judgement*), 496 (*Maxim*), 535 (*Affirmation*), 566 (*Phrase*), and 971 (*Condemnation*) in Roget. For example, the nominal part of Section 480 is,

> *480. Judgment. [Conclusion.]*
>
> *N. result, conclusion, upshot; deduction, inference, ergotism[Med]; illation; corollary, porism[obs3]; moral. estimation, valuation, appreciation, judication[obs3]; dijudication[obs3], adjudication; arbitrament, arbitrement[obs3], arbitration;*

*assessment, ponderation[obs3]; valorization. award, estimate; review, criticism, critique, notice, report. decision, determination, judgment, finding, verdict, **sentence**, decree; findings of fact; findings of law; res judicata[Lat]. plebiscite, voice, casting vote; vote &c. (choice) 609; opinion &c. (belief) 484; good judgment &c. (wisdom) 498. judge, umpire; arbiter, arbitrator; assessor, referee. censor, reviewer, critic;*

connoisseur; commentator &c. 524; inspector, inspecting officer. twenty-twenty hindsight [judgment after the fact]; armchair general, Monday morning quarterback.

Generally *sentence#n#*1 in WordNet can be projected into Section 496 and 566, and *sentence#n#*2 into Section 480 and 971, and *sentence#n#*3 into Section 535. With respect to the evaluation of Sim_{WN} in WordNet, *term* in Table 3 (a) is the hypernym of *sentence#n#*3; and *phrase* and *sentence#n#*1 distance themselves in three links, say, *sentence#n#*1 has a meronym of *clause* that is a coordinate of *phrase*; and *sentence#n#*2 bears the same hypernym with *verdict* within four links. Apart from the paradigmatic relationships in WordNet, the three words also connect with *sentence* through Sim_{RT} in Roget, where words such as *verdict* and *sentence*s are located under the same section—*Judgement* (480). However, *sentence* holds more relations of being in the same domain with its similar words in the thesaurus from $\mathbf{aN}_X$. For example, *penalty* and *sentence* come from/exist in Section 971, which expresses the notion of criminality deserving a penalty in a way of judicial sentence, and *prisoner* and *sentence* are situated in Section 971, which illustrates being in prison resulting from judgements in a court in the context of criminal law.

As we compute distributional similarity on the assumption of similar words sharing similar contexts conditioned by grammatical relations, in general more paradigmatic relations can be found than syntagmatic ones. In Table 4, the higher precision for WordNet than for Roget's Thesaurus show that distributionally similar words are more semantically similar rather than associated words. This is consistent with the conclusion of Kilgarriff and Yallop (2000) on computing distributional similarity that the hypothesis of similar words sharing similar contexts constrained by grammatical relations can yield *tighter* or WordNet-style thesauri, whereas the hypothesis of similar words sharing unconditioned co-occurrences can yield *looser* or Roget-style thesauri. Note that distributionally similar words could be semantically opposite to each other, given the common grammatical relations they often share. For example, in the automatic thesaurus produced with '***any two***', the nouns *failure* and *success*, or *strength* and *weakness*, are antonymous, as well the verbs *cry* and *laugh*, *deny* and *admit*.

It is clear that the 'gold standard' is subject to the vocabulary size of WordNet and Roget's Thesaurus. The worse case is from the 1911 version of Roget's Thesaurus we adopted, where words generated in modern times are not contained. For example words such as *software* and its distributionally similar words, including *emulator*, *unix*, *NT*, *Cobol*, *Oracle* (as the database system), *processor*, and *PC*, are not included in the 1911 version of Roget. We selected the target word with relatively higher frequencies in BNC and did a simple morphology analysis in the construction of the matrices using word-mapping table in WordNet, so that all nouns and verbs from automatic term clustering can be covered (at least in WordNet). However, not all word relationships in automatic thesauri could be contained in WordNet, even though we have included Roget to supply richer relationships. For example, take the words *sentence* and *detention*. In Table 3 (a) *detention* is listed in the top 20 similar words to *sentence* on $\mathbf{aN}_X$, but they have no direct or indirect links in WordNet, nor are they situated under any *topic* or *section* in Roget, but their intense association has become commonly used. Likewise, *kidnap* as one of the top 20 similar words to *attack* on $\mathbf{rV}_X$ in Table 3 (b), which is distributionally similar to *attack*, but there are no existing connections between them in WordNet and Roget.

7 Conclusion

With the introduction of grammatical relations in computing distributional similarity, automatic thesaurus construction can be improved through the interchangeability of similar words in diverse syntactically conditional contexts. Most methods still combined these contexts into one united representation for similarity computation, which worked analogously to these based on the premise of '*a bag of words*'. After the categorization of the syntactically conditioned contexts, through which similar words can be formed under the assumption of context interchangeability, automatic thesauri were yielded with significantly higher precision than the traditional methods. Future research will focus on clustering dependencies and extracting word senses from the thesaurus entries. Learning or enriching ontologies from automatic thesauri is also the next task.

8 References

Carroll, John, Ted Briscoe and Antonio Sanfilippo (1998). Parser Evaluation: a Survey and a New Proposal. In the First International Conference on Language Resources and Evaluation, 447-454. Granada, Spain.

Cruse, D. A. (1986). Lexical Semantics, Cambridge University Press.

Curran, James R. (2003). From Distributional to Semantic Similarity. Ph.D thesis

Deerwester, Scott C., Susan T. Dumais, Thomas K. Landauer, George W. Furnas and Richard A. Harshman (1990). Indexing by Latent Semantic Analysis. Journal of the American Society of Information Science 41(6): 391-407.

Fellbaum, Christiane (1998). WordNet: An Electronic Lexical Database. Cambridge, MA, The MIT Press.

Grefenstette, Gregory (1992). Sextant: Exploring Unexplored Contexts for Semantic Extraction from

Syntactic Analysis. In the 30th Annual Meeting of the Association for Computational Linguistics, 324-326. Newark, Delaware.

Grefenstette, Gregory (1993). Evaluation Techniques for Automatic Semantic Extraction: Comparing Syntactic and Window Based Approaches. In the Workshop on Acquisition of Lexical Knowledge from Text, 143-153.

Harris, Zellig (1985). Distributional Structure. In The Philosophy of Linguistics J. J. Katz, (ed). New York, Oxford University Press: 26-47.

Hindle, Donald (1990). Noun Classification from Predicate-argument Structures. In the 28th Annual Meeting of the Association for Computational Linguistics, 268-275. Pittsburgh, Pennsylvania.

Hirschman, Lynette, Ralph Grishman and Naomi Sager (1975). Grammatically-based Automatic Word Class Formation. Information Processing and Management 11: 39-57.

Kilgarriff, Adam (1997). I don't Believe in Word Senses. Computers and the Humanities 31(2): 91-113.

Kilgarriff, Adam (2004). How Dominant is the Commonest Sense of a Word? In the 7th International Conference (TSD 2004, Text, Speech and Dialogue), 103-112. Brno, Czech Republic.

Kilgarriff, Adam and Colin Yallop (2000). What's in a Thesaurus? In the Second International Conference on Language Resources and Evaluation, LREC-2000, 1371-1379. Athens, Greece.

Landauer, Thomas K. and Susan T. Dumais (1997). A Solution to Plato's Problem: the Latent Semantic Analysis Theory of Acquisition, Induction, and Representation of Knowledge. Psychological Review 104: 211-240.

Lin, Dekang (1997). Using Syntactic Dependency as a Local Context to Resolve Word Sense Ambiguity. In the 35th Annual Meeting of the Association for Computational Linguistics, 64-71. Madrid, Spain.

Lin, Dekang (1998). Automatic Retrieval and Clustering of Similar Words. In the 17th International Conference on Computational Linguistics, 768-774. Montreal, Quebec, Canada.

Lowe, Will (2001). Towards a Theory of Semantic Space. In the 23rd Annual Conference of the Cognitive Science Society, 576-581. Edinburgh, UK.

McCarthy, Diana, Rob Koeling, Julie Weeds and John Carroll (2004). Finding Predominant Senses in Untagged Text. In the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04), 267-287. Barcelona, Spain.

Miller, George A. and Walter G. Charles (1991). Contextual Correlates of Semantic Similarity. Language and Cognitive Processes 6(1): 1-28.

Molla, Diego and Ben Hutchinson (2003). Intrinsic versus Extrinsic Evaluations of Parsing Systems. In European Association for Computational Linguistics(EACL), workshop on Evaluation Initiatives in Natural Language Processing, 43-50. Budapest, Hungary.

Padó, Sebastian and Mirella Lapata (2007). Dependency-based construction of semantic space models. To appear in Computational Linguistics 33(2).

Pantel, Patrick and Dekang Lin (2002). Discovering Word Senses from Text. In the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 613-619. New York, NY, USA.

Resnik, Philip (1997). Selectional Preference and Sense Disambiguation. In ACL Siglex Workshop on Tagging Text with Lexical Semantics, Why, What and How?, 52-57. Washington, USA.

Sahlgren, Magnus (2006). The Word-Space Model: Using Distributional Analysis to Represent Syntagmatic and Paradigmatic Relations between Words in High-Dimensional Vector Spaces. Ph.D thesis

Salton, Gerard and Michael J. McGill (1986). Introduction to Modern Information Retrieval. New York, NY, USA, McGraw-Hill.

Schütze, Hinrich (1992). Dimensions of Meaning. In the 1992 ACM/IEEE Conference on Supercomputing, 787-796. Minneapolis, Minnesota, USA.

Sleator, Daniel and Davy Temperley (1991). Parsing English with a Link Grammar, Carnegie Mellon University.

Weeds, Julie Elizabeth (2003). Measures and Applications of Lexical Distributional Similarity. Ph.D thesis

Yang, Dongqiang and David M.W. Powers (2005). Measuring Semantic Similarity in the Taxonomy of WordNet. In the Twenty-Eighth Australasian Computer Science Conference (ACSC2005), 315-322. Newcastle, Australia, ACS.

Yang, Dongqiang and David M.W. Powers (2006). Verb Similarity on the Taxonomy of WordNet. In the 3rd International WordNet Conference (GWC-06), 121-128. Jeju Island, Korea.

Yarowsky, David (1993). One Sense per Collocation. In ARPA Human Language Technology Workshop, 266-271. Princeton, New Jersey.

Zipf, George Kingsley (1965). Human Behavior and the Principle of Least Effort: an Introduction to Human Ecology. N.Y., Hafner Pub. Co.

Relative Simulation and Model Checking of Real-Time Processes

Colin Fidge

Faculty of Information Technology,
Queensland University of Technology,
Brisbane, Queensland, Australia.
Email: c.fidge@qut.edu.au

Abstract

Simulation and model checking are commonly used to compare the behaviour of a computer-based system with its requirements specification. However, when upgrading an operational legacy system the challenge is usually to compare the behaviour of a proposed new system against an old trusted one. Doing this for time-sensitive control systems is awkward because the behaviour of the system is dependent on that of its physical environment. Consequently, the old and new systems can be compared meaningfully only when they are simulated under exactly the same conditions. In this paper we show how this can be done by simulating both the old and new systems simultaneously, with both system models linked to the same environment model. The resulting simulation traces and model checking counterexamples allow the behaviours of a legacy real-time system and its proposed replacement to be compared directly and easily.

Keywords: Simulation; Model checking; Real-time systems.

1 Introduction

Software upgrades are a serious maintenance issue for long-lived control systems. While major engineering assets such as coal draglines and aircraft have lifetimes measured in decades, the microprocessors and accompanying software embedded within them become obsolete within a few years. Techniques for safely replacing outmoded computer control systems with newer versions are the subject of considerable practical interest (Luke et al. 2001) and the process is governed by strict regulations worldwide (U.S. Federal Aviation Administration 2001).

A particular problem with upgrading such legacy systems is that the original requirements documents, system specifications and certification tests may no longer exist or may be irrelevant for a system which has evolved over a number of years. Instead, reliance is often placed on the old system having a 'clean' in-service history (Australian Defence Force 2003) so that the proposed upgrade can be assessed directly against the current system's behaviour.

Acknowledgements I wish to thank the anonymous reviewers for their suggested improvements to this paper. This work was funded by Australian Research Council Discovery-Projects grant DP0773012, *Rapidly Locating Items in Distribution Networks with Process-Driven Nodes*.

In particular, we are interested in formally modelling old and new real-time control systems so that their task-level behaviour can be compared using simulation and model checking. However, this process is difficult because the behaviour of embedded timing-dependent control systems is intimately linked to that of their physical environment. Therefore, an appropriate model of the (uncontrolled) external environment must be devised as well.

Furthermore, the behaviours of the old and new systems can be compared meaningfully only if they are simulated under exactly the same circumstances. It is not adequate, for instance, to run a random simulation of the old system followed by a simulation of the proposed new system if the model of the environment behaves differently in the two simulations.

An 'obvious solution' to this dilemma is to record a trace of the environment's behaviour during the first simulation and replay it during the second. Unfortunately, this is not possible in practice due to the inherently discrete nature of the simulations. For instance, consider a situation where we want to replace a legacy task running at a frequency of 5Hz with a new one running at 4Hz. The trace of the first task's simulation will include its interactions with the environment at times 200ms, 400ms, 600ms, 800ms, etc. The data in this trace is not useful when attempting to simulate the second task's interactions at times 250ms, 500ms, 750ms, etc. Creating environmental traces with events recorded at both tasks' frequencies is not possible either because there is typically a degree of 'jitter' (variability) in a task's periodicity, so the precise times at which it interacts with its environment cannot be predicted in advance.

In this paper we develop a solution to this problem which involves simulating the behaviours of both the old and new systems simultaneously, with both system models interacting independently with the same model of the environment. This allows the behaviours of the two systems, under exactly the same circumstances, to be compared directly. The simultaneous traces of the two systems' behaviours then provide an easy and intuitive way to see their similarities and differences. Model checking can also be used to explore the combined state space of the two systems and search for specific differences between them.

2 Previous Work

Our interest in this research is analysis of time-critical software tasks. A central concern with such tasks is whether or not they can be guaranteed to meet their deadlines when faced by preemption from higher-priority tasks and locking of shared resources by lower-priority tasks. Fortunately, real-time schedulability tests can be used to determine this, given certain timing characteristics of the task set (Buttazzo 1997). We take it for granted that such analyses have

already been done and that the task set's schedulability is not in question. Instead we are interested in the tasks' functional behaviour, an issue not addressed at all by scheduling theory.

Our approach is based on modelling the real-time tasks of interest and then analysing the models via simulation and model checking. There are, of course, numerous published case studies on modelling and analysis of time-critical processes. Behrmann et al. (2004) even cite several examples using the specific simulation toolkit that we use, including models of a gearbox controller, a power supply controller, an automobile supervisory controller, and so on. In each case, however, the analysis involves developing a model of a single system and either exploring its possible behaviours through simulation or confirming its adherence to formally-stated requirements through model checking. None of these prior examples address our concern of allowing two separate models to be directly compared with one another.

In fact, the approach closest to ours is the programming technique of 'relative' code debugging (Searle 2007). Relative debugging helps programmers locate errors by comparing the contents of key data structures within two executing programs, one program acting as a reference for the other. If the contents of the data structures differ at selected points during execution then there may be an error. The principle underlying this approach matches ours, but work in this area to date has not been devoted to real-time programs. Furthermore, our interest is in analysing systems at the tasking level, not at the level of individual program statements and data structures.

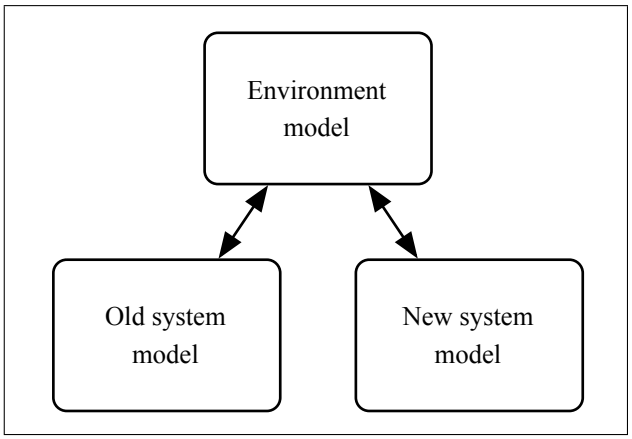


Figure 1: Overall structure of the simulations.

3 Approach

The fundamental idea underlying our approach is to link models of both the old control system and its proposed replacement to the same model of the physical environment (Figure 1). In this section we describe our formal models of periodic control tasks and their environment. As a motivating case study, we assume that the computational tasks of interest are part of an avionics system, and are meant to calculate the vertical veclocity of an aircraft via heights sampled periodically from the radar altimeter.

3.1 Experimental Environment

The models were developed using the UPPAAL real-time toolkit, which comprises an editor, simulator and model checker (Behrmann et al. 2004). UPPAAL allows systems to be modelled as sets of concurrent finite state automata augmented with integer-valued variables, special 'clock' variables, and synchronisation channels.

UPPAAL was chosen because it is optimised for analysis of time-dependent models (although our approach could equally well be applied in other simulation environments). In particular, UPPAAL's semantics assumes that all clock variables progress at the same rate, providing a form of implicit synchronisation between automata. Automata may share globally-declared variables. The timing semantics is that time progresses while an automaton resides in a particular state; transitions are instantaneous. States may be annotated with boolean expressions which must be true for the automaton to remain in that state. Transitions may be annotated with conditions that must be true for that transition to occur. Transitions may include variable assignments which are performed when the transition occurs. Transitions in concurrent automata may be explicitly synchronised via input and output events on shared channels, in which case the assignments performed by the 'output' transition are completed before those of the corresponding 'input' transition.

The experiments described in Section 4 below were conducted using UPPAAL Version 4.0.6 on an Apple MacBook laptop computer running the OS X operating system, Version 10.4.10. The simulation traces and model checking counterexamples were exported from UPPAAL as text files and filtered through a small 'awk' script written by the author to make them compatible with Apple's Grapher tool, Version 1.1. Figures 5 to 10 below were then all exported in Portable Document Format from Grapher.

3.1.1 Model of the Physical Environment

Simulating a real-time control system using finite state automata introduces the challenge of devising an appropriate discrete approximation of the system's continuous environment. For the particular case study of interest here we needed to simulate the altitude of an aircraft, taking into account its vertical velocity and acceleration.

In previous work we developed an environment model by treating each value of the observed variable's velocity as a distinct state (Fidge & Tian 2006). Here we use a different approach in which the velocity and acceleration are modelled as state variables. This produces a more concise automaton, but requires transitions to update a larger number of variables.

Figure 2 shows our UPPAAL model of the control task's physical environment. There is only a single state, **SensorIdle**. (UPPAAL's graphical user interface uses coloured text to distinguish different kinds of state and transition annotation. Here we use keywords **when**, **sync**, **while** and **do** for this purpose.)

As usual in a model checking tool, one of our primary modelling aims is to minimise the number of state changes, so a state transition occurs only when one of the old or new system tasks wishes to sample the aircraft's altitude, which it indicates by synchronising with channel *sample*. For each of the old and new system models there are three synchronised transitions, representing the cases where the aircraft's instantaneous acceleration increases, stays unchanged, or decreases, respectively.

Considering the middle transition in Figure 2, which represents the case where the acceleration is unchanged, we can see that the aircraft's instantaneous vertical velocity v is set to its previous value plus the instantaneous acceleration a multiplied by the difference between the time s at which the current altitude sample is being taken and the time p at which the last sample was taken (by either the old or new

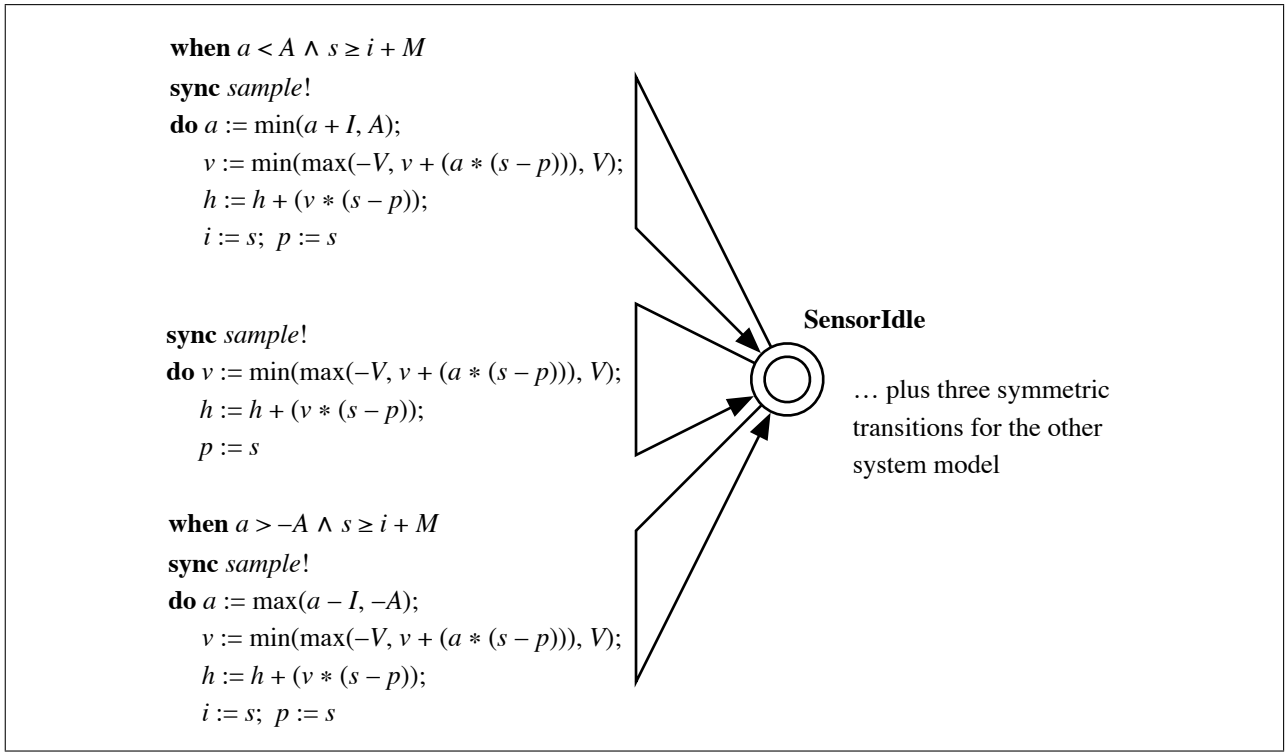


Figure 2: The state-machine model of the physical environment. Constants: A—Maximum possible acceleration (metres per second per second); V—Maximum possible velocity (metres per second); M—Minimum time allowed between increments of the acceleration (seconds); I—How much the acceleration changes in one increment (metres per second per second). Variables: h—Height (metres); v—Instantaneous velocity (metres per second); a—Instantaneous acceleration (metres per second per second); i—Time at which acceleration was last incremented (seconds); s—Time at which next altitude sample will be taken (seconds); p—Previous sample time (seconds). Variables h and s are shared with the tasks (Figure 4).

system models). Also, the velocity is bounded by constant V representing the aircraft's maximum possible vertical velocity (or at least the velocity within which the control system is required to function reliably). The aircraft's height h is then set to its previous value plus the velocity v times the difference between the current s and previous p sample times. Finally, the previous sample time p is set to the current sample time s in readiness for the next transition.

The uppermost transition in Figure 2 represents the case where the aircraft's vertical accleration increases. It is guarded by the requirement that the acceleration a is less than the maximum possible assumed acceleration A, and that the last sample time s is no less than the time i at which the acceleration last changed plus the minimum allowed separation between acceleration changes M. In this case the aircraft's instantaneous vertical acceleration a is increased by constant I (but is still bounded by maximum possible acceleration A). Also the time i at which the acceleration was last changed is set to the current sample time s. The aircraft's vertical velocity v, height h, and previous sample time p are updated as explained above.

Finally, the lowermost transition in Figure 2 models the case where the vertical acceleration decreases, and mirrors the uppermost transition. Although complicated, we have found that modelling the physical environment in this much detail produces reasonably 'smooth' behaviours and minimises modelling artifacts in the results. By changing the constants we can also simulate a wide range of dynamic behaviours.

Most importantly for our purposes, the model of the physical environment allows the aircraft's altitude to be sampled by either the model of the old system or its intended replacement. Thus, each of the three transitions described above is duplicated for the second system model. The only difference is that there is a separate '*sample*' synchronisation channel and a separate sample time variable 's' for each of the two control task models. In particular, since transitions are instantaneous in Uppaal, both system models can sample from the environment concurrently without interfering with one another.

3.1.2 Model of Control Tasks

Our models of the legacy and replacement control tasks all follow the same general structure. The challenge was to devise a model that captures the behaviour of a periodic control task which, at each invocation, takes a sample from the altitude sensor, uses this and one or more previous measurements to calculate the aircraft's vertical velocity, and then produces the calculated velocity as its output.

Taking our cue from real-time scheduling theory (Audsley et al. 1993), we assume that periodic task invocations are scheduled every T seconds and must complete their computation within a deadline of D seconds. We assume that the best-case (shortest) duration between the task sampling the altititude and producing the calculated vertical velocity is B seconds. (We also assume that a schedulability test has already been used to verify that each task invocation will finish by its deadline, so the task's worst-case execution time need not be modelled explicitly.)

Given these task characteristics, we modelled three extreme behaviours of interest, as shown in Figure 3. Case 1 represents the situation where the task invocation suffers no interference from other processor activity and completes its computation as early as possible. Case 2 describes the situation where the task completes all of its activities as late as possible, but still within its deadline. Case 3 models the maxi-

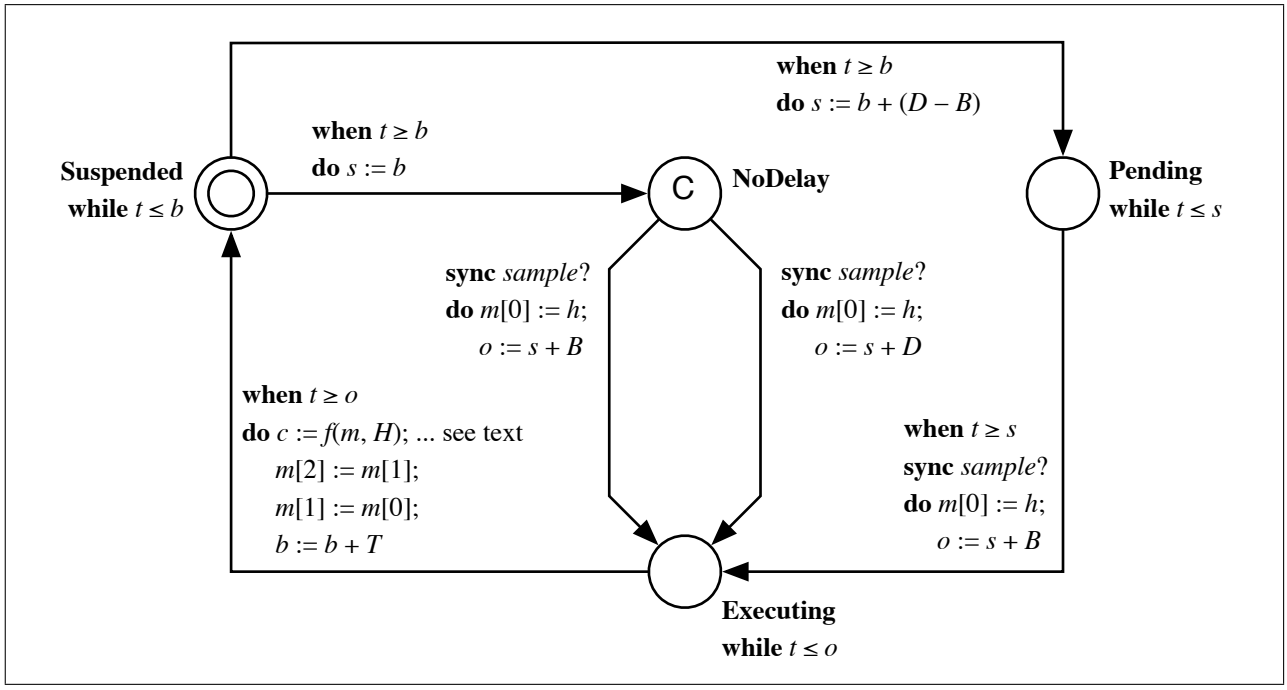


Figure 4: Model of a periodic task. Constants: T—Period (seconds); D—Deadline (seconds); B—Best-case execution time (seconds); H—Frequency (hertz). Variables: t—Current time (seconds); h—Height (metres); s—Time at which task takes next sample (seconds); c—Calculated velocity (metres per second); m—Measured altitudes (metres); b—Beginning time of the next task invocation (seconds); o—Time at which task produces its output (seconds).

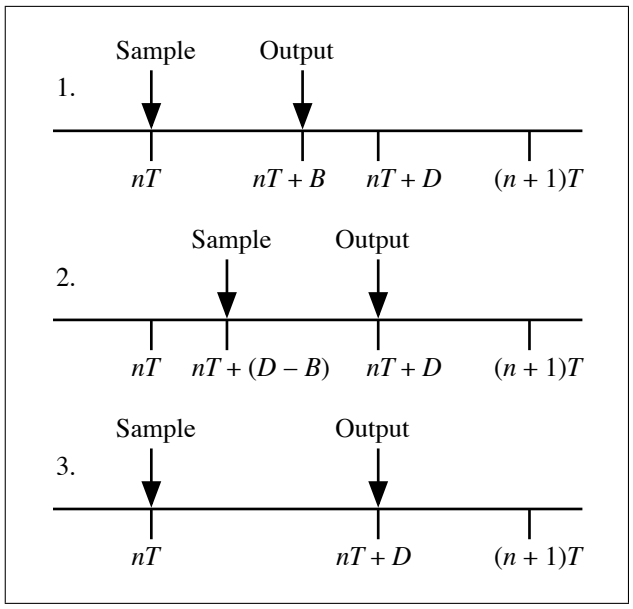


Figure 3: Timelines showing the three behaviours modelled for the nth invocation of a task. Constants: T—Period; D—Deadline; B—Best-case execution time.

mum possible separation between the task sampling its input and producing its output. Although there are other possible task behaviours between these extremes, these cases are sufficient for comparing the *worst-case* behaviours of each task.

To capture these task behaviours in UPPAAL we developed the state machine model shown in Figure 4. State **Suspended** represents the situation where the task is waiting for its next invocation. (The nested circle indicates that this is the initial state.) This state persists while the current time t does not exceed the task invocation's beginning time b. The three paths leading from state **Suspended** back to itself model the three behaviours described in Figure 3. All transitions leading from state **Suspended** are guarded by the requirement that current time t is not less than the task's beginning time b. Combined with the constraint on the state, this forces the chosen transition to occur at time b exactly.

The uppermost path in Figure 4 models the case where the task invocation performs all of its actions as late as possible (Case 2 in Figure 3). When state **Suspended** is left the time s at which the task samples the aircraft's altitude is set to the latest possible time, and the task enters state **Pending** until this time. When state **Pending** is left the task synchronises with the environment model, via channel *sample*, and stores a measurement of the aircraft's height h in array m. The model also sets the time o at which the task invocation will produce its output, and enters state **Executing** which represents the situation where the task is calculating the aircraft's vertical velocity from recently sampled altitude readings.

When state **Executing** is left, at time o, the task's calculated velocity c is produced as a function f of the array m of altitude measurements and the task's frequency H. The choice of function f depends on the type of task being modelled (see Section 4). In addition, the most recent altitude measurements are shifted along array m and the beginning time b for the next task invocation is calculated as the last invocation's beginning time plus the task's period T.

The other two paths through Figure 4, via state **NoDelay**, model Cases 1 and 3 from Figure 3. In both cases the task invocation samples the aircraft's altitude immediately. (In UPPAAL a 'C' within a state indicates that this is a 'committed' transition which occurs straight away.) From state **NoDelay** the only difference between these two cases is the time o at which the subsequent output will be produced.

Although simple, the results in Section 4 below show that this model accurately captures a periodic task's dynamic behaviour.

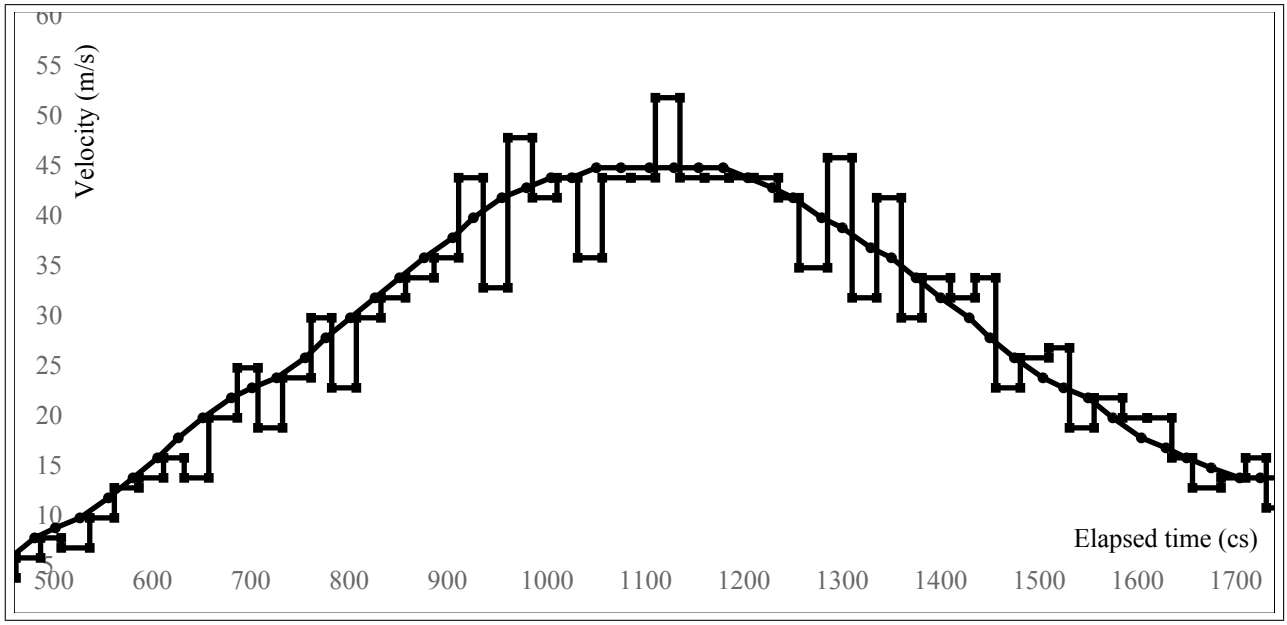


Figure 6: Effect of (worst-case) jitter on a simple task. A simulation showing the task's calculated velocity (the step function) against the environment's instantaneous velocity (smooth curve). Task constants: $T = 250$ms, $D = 100$ms, $B = 60$ms.

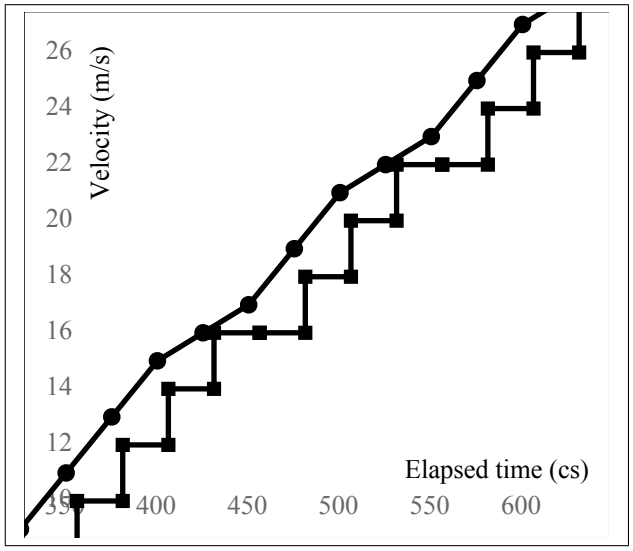


Figure 5: Velocity calculated by a simple task (the step function) compared with the instantaneous velocity used by the environment model (smooth curve) with no task jitter. Task constants: $T = 250$ms, $D = 60$ms, $B = 60$ms.

4 Experimental Results

In this section we present the outcomes of a number of experiments conducted using the models described in Section 3. In particular, we show how our approach of simulating both the old and new systems simultaneously, interacting with the same environment, allows their respective behaviours to be compared easily.

The same environment model was used for all of the experiments shown below. With respect to Figure 2, the environmental constants used in the experiments were a maximum vertical acceleration A of 10m/s^2, a maximum vertical velocity V of 140m/s, a fixed increment I to the acceleration when it changes of 3m/s^2, and a minimum allowed time M between changes to the acceleration of 500ms.

4.1 Validating the Task Model

We firstly need to confirm that our models of periodic control tasks and their environment reflect the actual dynamic behaviour of such systems.

As an initial test, Figure 5 shows a simulation of a simple task which calculates the aircraft's vertical velocity based on the last two measured altitudes. With respect to function f in Figure 4, the calculated velocity c was produced from the task's frequency H and array m of measured altitudes as follows.

$$c \;:=\; (m[0] - m[1]) * H$$

Figure 5 clearly shows the calculated velocity lagging behind the 'true' instantaneous velocity, which is climbing rapidly at this point in the simulation, as we would expect due to the time required to sample altitudes and perform the calculation. (The horizontal axis in this and subsequent figures is measured in centiseconds, i.e., hundredths of a second. The reason for this unusual choice of time unit is discussed in Section 5.)

The task's deadline D and best-case execution time B were the same in this experiment, so there was no jitter (variability) in the periodicity of times at which the task sampled altitudes or produced velocities. In practice, however, the execution of most periodic tasks is disturbed by either preemption from other software tasks running on the processor or interruptions from hardware events. To model this situation we ran another experiment in which the task's deadline D was increased. This allowed variability in the times at which each task invocation performs its actions, as per Figure 3.

The results of this experiment are shown in Figure 6. This time the task's calculated behaviour fails to follow the instantaneous velocity closely. Furthermore, as the velocity increases, so do the errors in the calculated velocity.

In fact, this is exactly the behaviour we expect from a task with appreciable jitter. The expression above for calculating velocity c assumes that consecutive altitude measurements are taken exactly $1/H$ seconds apart. However, if the actual samples are taken closer together than this, due to jitter in the

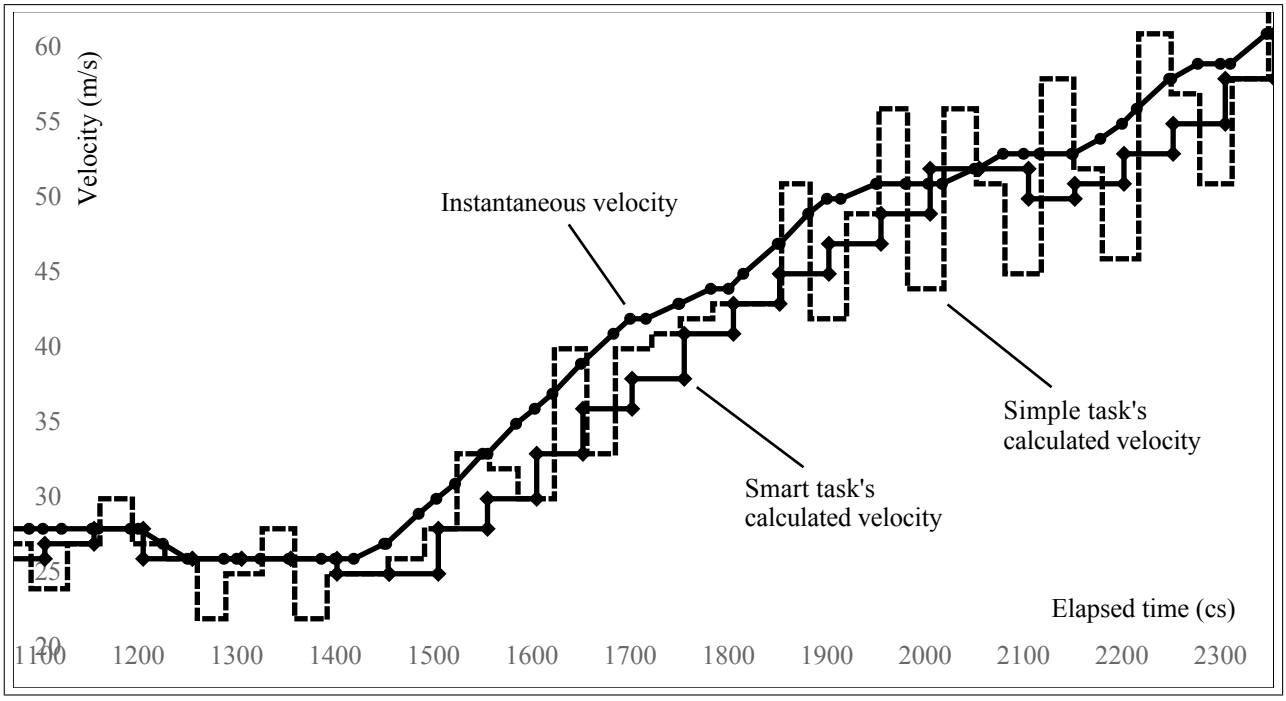


Figure 7: Comparison of a fast simple task (dashed line) with a slower but smarter one (solid line). Simple task constants: $T = 333$ms, $D = 60$ms, $B = 20$ms. Smart task constants: $T = 500$ms, $D = 50$ms, $B = 20$ms.

task's periodicity, then the velocity will be underestimated. If samples are taken further apart than expected then the velocity will be overestimated. These effects account for the way the task both under and overestimates the calculated velocity in our simulation, and the fact that the magnitude of the error is proportional to the absolute velocity.

For instance, in Figure 6 samples could be taken as far apart as $T + (D - B) = 250 + (100 - 60) = 290$ms. This would skew the results by a factor of $290/250 = 1.16$, meaning that when the true velocity is, for example, 45m/s, the simple task may report it as 52.2m/s. Indeed, we see this occur at the top of the curve in Figure 6 where, even though the instantaneous velocity is constant, the calculated velocity jumps upwards. (The effects of jitter would not normally be as severe as shown here, but keep in mind that we are modelling worst-case behaviours.)

From these and other experiments we concluded that our task and environment models do indeed capture the dynamic system properties of interest.

4.2 Comparing Different Calculations

Most importantly, however, by constructing our models as shown in Figure 1, we can simulate two different tasks under exactly the same conditions to compare their respective behaviours.

For example, having observed the erratic behaviour of the simple task in Section 4.1, we may consider whether we can improve its (worst-case) behaviour by changing the way the velocity is calculated. One way of doing this would be to calculate the velocity using the last three measured altitudes, rather than just the last two, using an average value as follows.

$$c := \frac{((m[0] - m[1]) + (m[1] - m[2])) * H}{2}$$

Figure 7 then shows the result of an experiment to directly compare the behaviour of the original simple task and this proposed new 'smart' one, both suffering from worst-case jitter. Furthermore, in this particular example the simple task runs at a frequency of 3Hz, while the smart task runs at only 2Hz, thus putting the smart task at a significant disadvantage. Around 18 seconds into the simulation, for instance, it is obvious that the smart task's output lags further behind the instantaneous velocity than that of the simple task, because the simple task's output is updated more often.

Nevertheless, the figure clearly shows that the different calculation in the smart task produces a smoother result, despite this handicap. As the true velocity increases the simple task's calculated velocity becomes unstable, whereas the smart task continues to follow the true velocity closely. (In the particular experiment shown in Figure 7 the smart task has a shorter deadline than the simple one and thus suffers less from jitter. Nevertheless other experiments showed that the relative 'smoothness' of the smart task's behaviour is primarily due to the averaging effect of the way it calculates velocities.)

Simulating the behaviour of the two tasks together in this way thus allows us to compare various aspects of their behaviours quickly and easily. By varying the task constants we can also calibrate them with respect to one another.

4.3 Comparing Different Task Sets

We can also use our approach to compare not just single tasks but different task sets. For instance, a common architecture in avionics control systems is to have an autonomous Input-Output Processor running in parallel with the Central Processing Unit (Falardeau 1994). This allows the CPU to continue calculating while the IOP interacts with hardware devices to get sensor data.

Therefore, we decided to conduct an experiment which compared the behaviour of a single dedicated task to an IOP-CPU task pair. We assumed that the vertical velocity calculation is split into two halves. A periodic task on the Input-Output Processor samples altitudes and puts the values into a shared memory location, and a periodic task in the Central Processing Unit reads these values and uses them to calculate the vertical velocity. Both task models were based

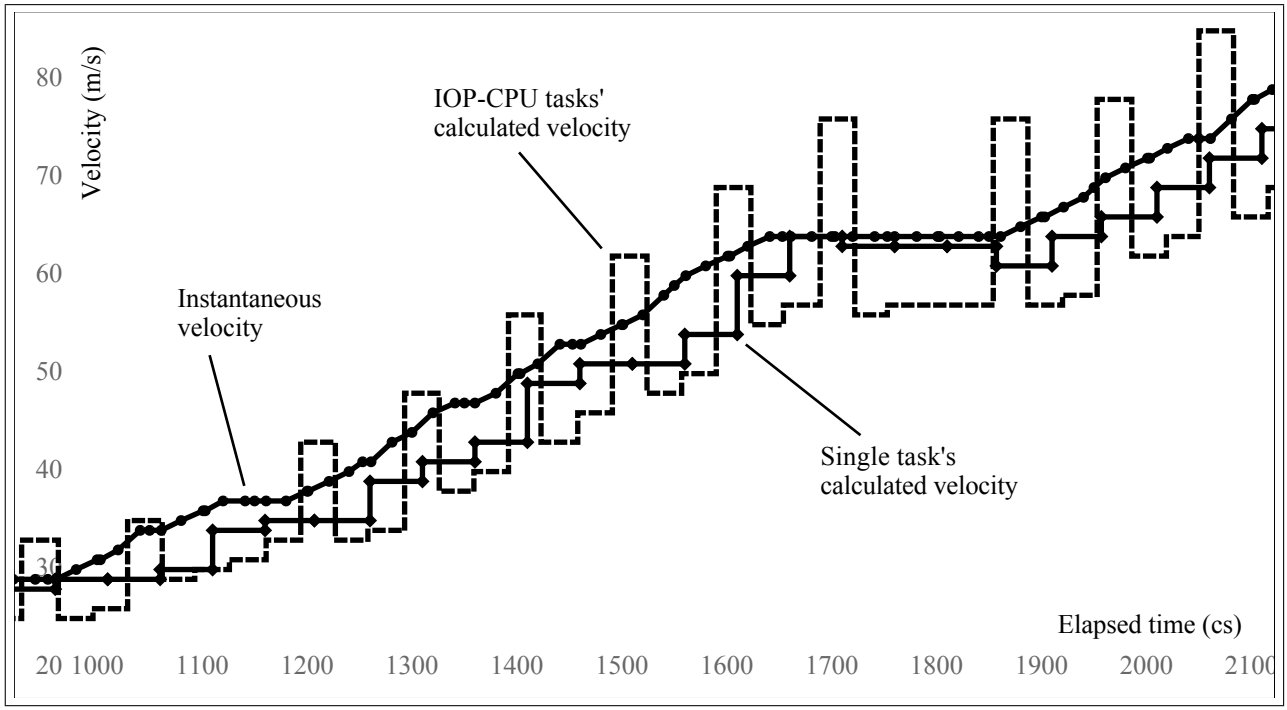


Figure 8: Comparison of a single task with a combination of an IOP task and a CPU task. Single task constants: $T = 500$ms, $D = 100$ms, $B = 70$ms. IOP task constants: $T = 200$ms, $D = 40$ms, $B = 30$ms. CPU task constants: $T = 333$ms, $D = 60$ms, $B = 40$ms.

on the one in Figure 4. The IOP task synchronised with the environment via a *sample* channel in the usual way and put the sampled altitude into a shared variable. However, the CPU task was slightly simpler than Figure 4 because it did not need to synchronise and just read from the global variable shared with the IOP task at appropriate times. (At the tasking level the delay caused by code being blocked awaiting access to the shared variable is accounted for in the task's timing constants.)

We then conducted an experiment to compare a single task with an IOP-CPU task combination. Both the single task and the CPU task used the 'smart' velocity calculation from Section 4.2. For the purposes of comparison the single task's execution time was split between the IOP and CPU tasks. However, both the IOP and CPU tasks were given shorter periods than the single task. Also their periods were different from one another. The IOP task was scheduled at a frequency of 5Hz, while the CPU task ran at only 3Hz. This was done on the assumption that it would allow the CPU task to always find a reasonably 'fresh' altitude stored in the memory location shared with the IOP task, so that we should get a smooth calculated velocity.

However, as shown in Figure 8, the IOP-CPU tasks performed worse than the single task, despite the single task running at only 2Hz. The highly erratic behaviour of the IOP-CPU combination was caused by the difference in their frequencies which created an extreme kind of jitter across successive task invocations. Furthermore, we also recognised from this result that the velocity calculation in the CPU task, which was based on its own frequency, was inadequate in this situation since it also needs to take into account the delay introduced by the IOP task. This explains the asymmetry in the IOP-CPU tasks' calculated velocity.

To solve this, we realised that there needs to be a harmonic relationship in the communicating tasks' periods. To confirm this hypothesis we changed the CPU task's period to match that of the IOP task. We also introduced an initial 100ms offset in the CPU task's schedule, so that CPU task invocations never overlap in time with those of the IOP task. (In practice this would allow us to avoid the need for a mutual-exclusion lock on the variable shared by the two tasks.)

Rerunning the experiment then produced a dramatically different result as shown in Figure 9. Not only does the IOP-CPU pair now produce a smooth calculated velocity, but thanks to their higher (shared) frequency they outperform the single task. It is clear from the figure that the IOP-CPU tasks' calculated velocity stays closer to the true velocity than that of the single task.

Again, these experiments show how the model allows us to compare different possible implementations easily and to identify design problems even before the corresponding programs are written.

4.4 Searching for Differences

Of course, the real advantage of using a tool such as UPPAAL is that as well as exploring random or user-guided behaviours via simulation, we can also use model checking to search for particular states of interest. Recall that model checking involves an exhaustive state-space search to locate specific states expressed through temporal logic formulæ (Clarke & Schlingloff 1999). In our application this means that we can use UPPAAL's model checker to search for significant differences in the behaviour of an old system and its intended replacement.

As an example, consider the situation where as part of a software upgrade we need to change the tasking schedule. The original task was invoked frequently but had high jitter. The question is whether it would be acceptable to replace it with a task that is invoked less frequently but which has a lower degree of jitter. (Both tasks calculate velocities as per the 'smart' task described in Section 4.2.)

In this case we may wish to check the following temporal logic safety property, where c is the more frequent (4Hz) task's calculated velocity and c' is the

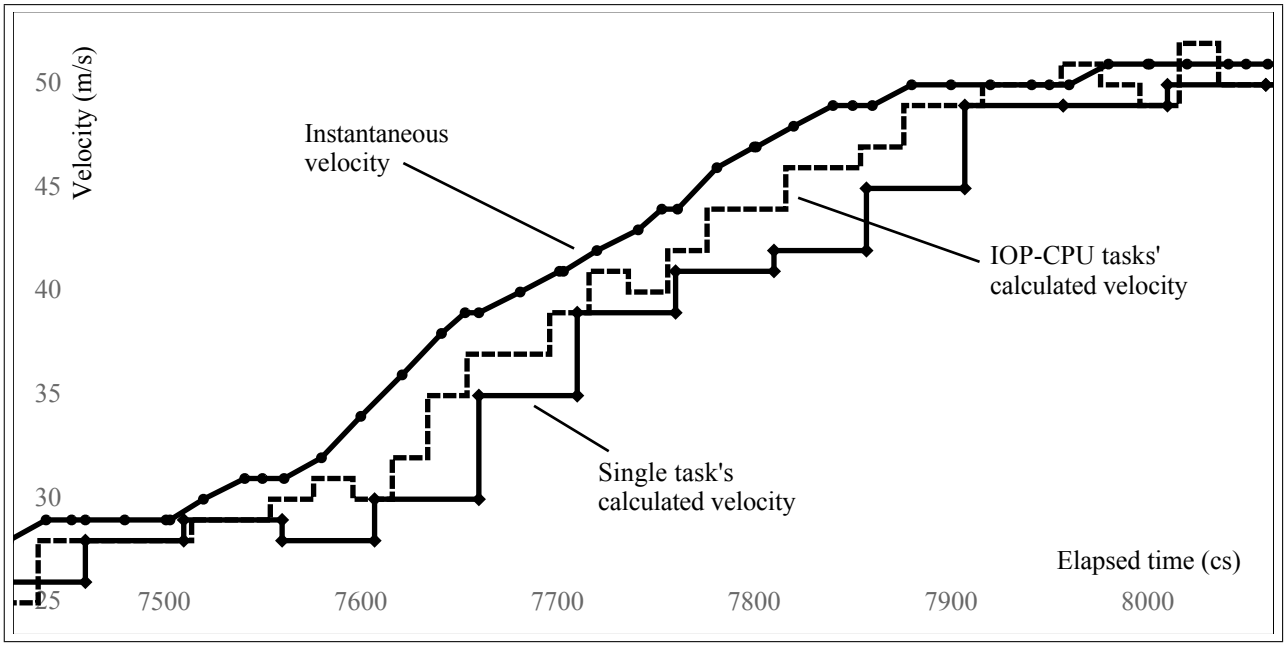


Figure 9: Comparison of a single task with an improved IOP-CPU task combination. Single task constants: $T = 500$ms, $D = 100$ms, $B = 70$ms. IOP task constants: $T = 200$ms, $D = 40$ms, $B = 30$ms. CPU task constants: $T = 200$ms, $D = 60$ms, $B = 40$ms and initial offset 100ms.

less frequent (2Hz) task's calculated velocity.

$$\Box(c - 6 \leq c' \leq c + 6)$$

In other words, we want to confirm that the replacement task's calculated velocity c' always stays within 6m/s of the original task's calculation c.

In this case UPPAAL's model checker quickly produced a counterexample to prove that this property does not hold. Figure 10 shows the resulting trace. The particular situation found is one where the velocity is decreasing rapidly and at a high negative velocity the difference in the two task's velocities exceeds 6m/s.

Interestingly, although a large difference was found between the original and proposed replacement tasks' behaviours, this does not necessarily suggest that the replacement task is not adequate. In fact, the primary cause of the difference here is the jitter in the *original* task. The figure shows that the less-frequently invoked replacement task (dashed line) has a comparatively smooth behaviour, so we may still choose to use it in place of the original task.

4.5 Failing to Find Differences

As a final experiment we chose to refine our model checking result. In the particular counterexample in Figure 10 the large difference between the two tasks' calculated vertical velocities occurs in the circumstance of a high negative velocity exceeding -90m/s. We may therefore ask whether such a difference could ever occur under less extreme conditions.

The following safety property asserts that the replacement task's calculated velocity c' always stays within 6m/s of the original task's calculation c, provided that the original task's velocity does not exceed an absolute value of 70m/s.

$$\Box((-70 \leq c \leq 70) \Rightarrow (c - 6 \leq c' \leq c + 6))$$

Attempting to check this property using UPPAAL's model checker produced inconclusive results, however. Trying depth-first or 'random depth-first' searches resulted in immediate integer overflows. This is not surprising given the way we have expressed the models, because the timestamp and altitude variables in the model can grow without bound.

On the other hand a breadth-first search became hopelessly memory-bound, attempting to maintain the enormous tree of states visited, despite the powerful optimisation techniques used by UPPAAL (Larsen et al. 1997). Again this is not surprising given that this particular model has numerous integer-valued variables, thus resulting in an enormous state space.

Ultimately, therefore, all we can conclude is that it appears highly likely that the safety property holds, since no counterexample could be found within reasonable time and space limits. However, it is an inevitable characteristic of models such as ours, with unbounded integer-valued variables, that the entire state space cannot be searched.

5 Discussion

The UPPAAL toolkit provided an extremely convenient environment in which to conduct this research. In particular, the implicitly synchronised clocks supported by its timed automata semantics allowed us to express independent models of the old and new systems which nonetheless both progressed at the same rate, and the instantaneous transitions in UPPAAL's timed automata semantics allowed the two system models to access the same environment concurrently without interfering with one another.

A weakness of UPPAAL, however, is its limited support for arithmetic. The current version supports 16-bit integers only. Working with such a limited range of values was awkward. Our first attempted models used centimetres as the unit for lengths and milliseconds as the unit for time, as these were natural choices for the problem domain. However, these units were so small that the simulations failed with integer overflows after only a few seconds of simulated time. We were forced, therefore, to re-express our models using larger units, specifically decimetres for lengths and centiseconds for times. This complicated the arithmetic in the models somewhat because it became necessary to scale the results to more commonly-used

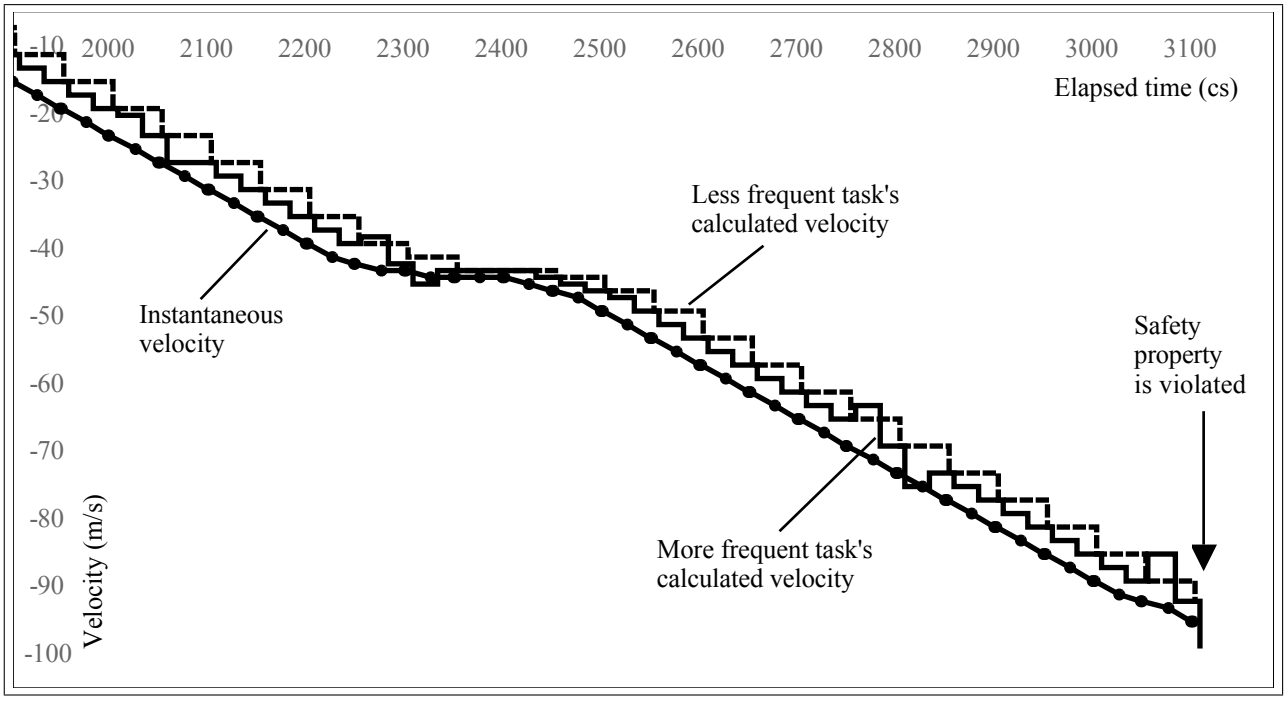


Figure 10: Model checking counterexample showing how a 6m/s difference can occur between two task's calculated velocities. More frequent task's constants: $T = 250$ms, $D = 100$ms, $B = 70$ms. Less frequent task's constants: $T = 500$ms, $D = 50$ms, $B = 40$ms.

units. (For clarity, the arithmetic needed to do this has been excised from Figures 2 and 4.)

6 Conclusion

The dynamic behaviour of real-time control tasks is complex and is intimately linked to that of their physical environment. In control system maintenance the behaviour of a proposed new system can be compared meaningfully with that of an existing legacy system only within the same environment. Here we have shown how this can be done in a real-time simulation and model checking toolkit by modelling both old and new systems concurrently with a shared environment model. The environment was accessed by each of the system models in a way that prevented the two models from interfering with one another. This allowed the separate behaviour of the two systems to be compared directly. Pleasingly, a variety of experiments showed that the outcomes are both easy to interpret, and match our intuitive understanding of the dynamic processes involved.

References

Audsley, N., Burns, A., Richardson, M., Tindell, K. & Wellings, A. (1993), 'Applying new scheduling theory to static priority pre-emptive scheduling', *Software Engineering Journal* **8**(5), 284–292.

Australian Defence Force (2003), *Airworthiness Design Requirements Manual*. Australian Air Publication 7001.054(AM1).

Behrmann, G., David, A. & Larsen, K. G. (2004), A tutorial on UPPAAL, Technical report, Department of Computer Science, Aalborg University.

Buttazzo, G. C. (1997), *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, Kluwer.

Clarke, E. M. & Schlingloff, B.-H. (1999), Model checking, *in* A. Robinson & A. Voronkov, eds, 'Handbook of Automated Reasoning', Elsevier.

Falardeau, J. D. G. (1994), Schedulability analysis in rate monotonic based systems with application to the CF-188, Master's thesis, Department of Electrical and Computer Engineering, Royal Military College of Canada.

Fidge, C. J. & Tian, Y.-C. (2006), Functional analysis of a real-time protocol for networked control systems, *in* S. Graf & W. Zhang, eds, 'Proceedings of the Fourth International Symposium on Automated Technology for Verification and Analysis (ATVA 2006)', Vol. 4218 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 446–460.

Larsen, K. G., Larsson, F., Pettersson, P. & Yi, W. (1997), Efficient verification of real-time systems: Compact data structure and state space reduction, *in* 'Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS'97)', IEEE Computer Society, pp. 14–24.

Luke, J. A., Haldeman, D. G. & Cannon, W. J. (2001), 'A COTS-based replacement strategy for aging avionics computers', *CrossTalk—The Journal of Defense Software Engineering* pp. 14–17.

Searle, A. (2007), Automatic Relative Debugging, PhD thesis, Faculty of Information Technology, Queensland University of Technology.

U.S. Federal Aviation Administration (2001), *Guidelines for the Approval of Software Changes in Legacy Systems Using RTCA DO-178B*. FAA Notice N8110.89.

Author Index

Recent Volumes in the CRPIT Series

ISSN 1445-1336

Listed below are some of the latest volumes published in the ACS Series *Conferences in Research and Practice in Information Technology*. The full text of most papers (in either PDF or Postscript format) is available at the series website `http://crpit.com`.

Volume 67 - Conceptual Modelling 2007
Edited by John F. Roddick, *Flinders University* and Annika Hinze, *University of Waikato, New Zealand.* January, 2007. 978-1-920682-48-4.
Contains the proceedings of the Fourth Asia-Pacific Conference on Conceptual Modelling (APCCM2007), Ballarat, Victoria, Australia, January 2007.

Volume 68 - ACSW Frontiers 2007
Edited by Ljiljana Brankovic, *University of Newcastle*, Paul Coddington, *University of Adelaide*, John F. Roddick, *Flinders University*, Chris Steketee, *University of South Australia*, Jim Warren, *the University of Auckland*, and Andrew Wendelborn, *University of Adelaide.* January, 2007. 978-1-920682-49-1.
Contains the proceedings of the ACSW Workshops - The Australasian Information Security Workshop: Privacy Enhancing Systems (AISW), the Australasian Symposium on Grid Computing and Research (AUSGRID), and the Australasian Workshop on Health Knowledge Management and Discovery (HKMD), Ballarat, Victoria, Australia, January 2007.

Volume 69 - Safety Critical Systems and Software 2006
Edited by Tony Cant, *Defence Science and Technology Organisation, Australia.* February, 2007. 978-1-920682-50-7.
Contains the proceedings of the 11th Australian Conference on Safety Critical Systems and Software, August 2006, Melbourne, Australia.

Volume 70 - Data Mining and Analytics 2007
Edited by Peter Christen, Paul Kennedy, Jiuyong Li, Inna Kolyshkina and Graham Williams. December, 2007. 978-1-920682-51-4.
Contains the proceedings of the 6th Australasian Data Mining Conference (AusDM 2007), Gold Coast, Australia. December 2007.

Volume 72 - Advances in Ontologies 2006
Edited by Mehmet Orgun *Macquarie University* and Thomas Meyer, *National ICT Australia, Sydney.* December, 2006. 978-1-920682-53-8.
Contains the proceedings of the Australasian Ontology Workshop (AOW 2006), Hobart, Australia, December 2006.

Volume 73 - Intelligent Systems for Bioinformatics 2006
Edited by Mikael Boden and Timothy Bailey *University of Queensland.* December, 2006. 978-1-920682-54-5.
Contains the proceedings of the AI 2006 Workshop on Intelligent Systems for Bioinformatics (WISB-2006), Hobart, Australia, December 2006.

Volume 74 - Computer Science 2008
Edited by Gillian Dobbie, *University of Auckland, New Zealand* and Bernard Mans *Macquarie University.* January, 2008. 978-1-920682-55-2.
Contains the proceedings of the Thirty-First Australasian Computer Science Conference (ACSC2008), Wollongong, NSW, Australia, January 2008.

Volume 75 - Database Technologies 2008
Edited by Alan Fekete, *University of Sydney* and Xuemin Lin, *University of New South Wales.* January, 2008. 978-1-920682-56-9.
Contains the proceedings of the Nineteenth Australasian Database Conference (ADC2008), Wollongong, NSW, Australia, January 2008.

Volume 76 - User Interfaces 2008
Edited by Beryl Plimmer and Gerald Weber *University of Auckland.* January, 2008. 978-1-920682-57-6.
Contains the proceedings of the Ninth Australasian User Interface Conference (AUIC2008), Wollongong, NSW, Australia, January 2008.

Volume 77 - Theory of Computing 2008
Edited by James Harland, *RMIT University* and Prabhu Manyem, *University of Ballarat.* January, 2008. 978-1-920682-58-3.
Contains the proceedings of the Fourteenth Computing: The Australasian Theory Symposium (CATS2008), Wollongong, NSW, Australia, January 2008.

Volume 78 - Computing Education 2008
Edited by Simon, *University of Newcastle* and Margaret Hamilton, *RMIT University.* January, 2008. 978-1-920682-59-0.
Contains the proceedings of the Tenth Australasian Computing Education Conference (ACE2008), Wollongong, NSW, Australia, January 2008.

Volume 79 - Conceptual Modelling 2008
Edited by Annika Hinze, *University of Waikato, New Zealand* and Markus Kirchberg, *Massey University, New Zealand.* January, 2008. 978-1-920682-60-6.
Contains the proceedings of the Fifth Asia-Pacific Conference on Conceptual Modelling (APCCM2008), Wollongong, NSW, Australia, January 2008.

Volume 80 - Health Data and Knowledge Management 2008
Edited by James R. Warren, Ping Yu, John Yearwood and Jon D. Patrick. January, 2008. 978-1-920682-61-3.
Contains the proceedings of the Australasian Workshop on Health Data and Knowledge Management (HDKM 2008), Wollongong, NSW, Australia, January 2008.

Volume 81 - Information Security 2008
Edited by Ljiljana Brankovic, *University of Newcastle* and Mirka Miller, *University of Ballarat.* January, 2008. 978-1-920682-62-0.
Contains the proceedings of the Australasian Information Security Conference (AISC 2008), Wollongong, NSW, Australia, January 2008.

Volume 82 - Grid Computing and e-Research
Edited by Wayne Kelly and Paul Roe *QUT.* January, 2008. 978-1-920682-63-7.
Contains the proceedings of the Australasian Workshop on Grid Computing and e-Research (AusGrid 2008), Wollongong, NSW, Australia, January 2008.

Volume 83 - Challenges in Conceptual Modelling
Edited by John Grundy, *University of Auckland, New Zealand*, Sven Hartmann, *Massey University, New Zealand*, Alberto H.F. Laender, *UFMG, Brazil*, Leszek Maciaszek, *Macquarie University, Australia* and John F. Roddick, *Flinders University, Australia.* December, 2007. 978-1-920682-64-4.
Contains the tutorials, posters, panels and industrial contributions to the 26th International Conference on Conceptual Modeling - ER 2007.

Volume 84 - Artificial Intelligence and Data Mining 2007
Edited by Kok-Leong Ong, *Deakin University, Australia,* Wenyuan Li, *University of Texas at Dallas, USA* and Junbin Gao, *Charles Sturt University, Australia.* December, 2007. 978-1-920682-65-1.
Contains the proceedings of the 2nd International Workshop on Integrating AI and Data Mining (AIDM 2007), Gold Coast, Australia. December 2007.

Volume 86 - Safety Critical Systems and Software 2007
Edited by Tony Cant, *Defence Science and Technology Organisation, Australia.* December, 2007. 978-1-920682-67-5.
Contains the proceedings of the 12th Australian Conference on Safety Critical Systems and Software, August 2006, Adelaide, Australia.