

CONFERENCES IN RESEARCH AND PRACTICE IN  
INFORMATION TECHNOLOGY

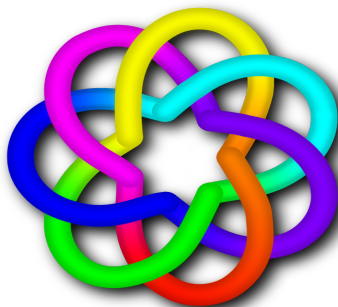
VOLUME 102

# COMPUTER SCIENCE 2010

AUSTRALIAN COMPUTER SCIENCE COMMUNICATIONS, VOLUME 32, NUMBER 1



AUSTRALIAN  
COMPUTER  
SOCIETY



 **CORE**  
*Computing Research & Education*



# COMPUTER SCIENCE 2010

Proceedings of the  
Thirty-Third Australasian Computer Science Conference  
(ACSC 2010), Brisbane, Australia,  
January 2010

Bernard Mans and Mark Reynolds, Eds.

Volume 102 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 2010.** Proceedings of the Thirty-Third Australasian Computer Science Conference (ACSC 2010), Brisbane, Australia, January 2010

**Conferences in Research and Practice in Information Technology, Volume 102.**

Copyright ©2010, Australian Computer Society. Reproduction for academic, not-for-profit purposes permitted provided the copyright text at the foot of the first page of each paper is included.

Editors:

**Bernard Mans**

Department of Computing  
Division of Information and Communication Sciences  
Macquarie University  
Sydney, NSW 2109  
Australia  
Email: [bmans@ics.mq.edu.au](mailto:bmans@ics.mq.edu.au)

**Mark Reynolds**

School of Computer Science and Software Engineering  
Faculty of Engineering, Computing and Mathematics  
The University of Western Australia  
Crawley, WA 6009  
Australia  
Email: [mark@csse.uwa.edu.au](mailto:mark@csse.uwa.edu.au)

Series Editors:

Vladimir Estivill-Castro, Griffith University, Queensland  
Simeon J. Simoff, University of Western Sydney, NSW

[crpit@scm.uws.edu.au](mailto:crpit@scm.uws.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 102.  
ISSN 1445-1336.  
ISBN 978-1-920682-83-5.

Printed, December 2009 by UWS Press, Locked Bag 1797, South Penrith DC, NSW 1797, Australia  
Document engineering by Susan Henley, University of Western Sydney  
Cover Design by Matthew Brecknell, Queensland University of Technology  
CD Production by FATS Digital, 318 Montague Road, West End QLD 4101, <http://www.fats.com.au/>

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



## Table of Contents

### Proceedings of the Thirty-Third Australasian Computer Science Conference (ACSC 2010), Brisbane, Australia, January 2010

Preface .....	vii
Programme Committee .....	viii
Organising Committee .....	ix
Welcome from the Organising Committee .....	x
CORE - Computing Research & Education .....	xi
ACSW Conferences and the Australian Computer Science Communications .....	xii
ACSW and ACSC 2010 Sponsors .....	xiv

### Contributed Papers

An Empirical Study of Overriding in Open Source Java .....	3
<i>Ewan Tempero, Steve Counsell and James Noble</i>	
Modular Transactional Memory .....	13
<i>Matthew Brecknell and Paul Roe</i>	
Memory Efficient State-Space Analysis in Software Model-Checking .....	23
<i>Anshuman Mukherjee, Zahir Tari and Peter Bertok</i>	
Efficient Contour Line Labelling for Terrain Modelling .....	33
<i>Burkhard Wuensche and Xin Xie</i>	
Measuring Visual Consistency in 3D Rendering Systems .....	43
<i>Alfredo Nantes, Ross Brown and Frederic Maire</i>	
Fluid Dynamic Visualisations of Cuttings-Bleeding for Virtual Reality Heart Beating Surgery Simulation .....	53
<i>Sugeng Rianto and Ling Li</i>	
Improved Consensus Clustering via Linear Optimization .....	61
<i>Nicholas Downing, Anthony Wirth and Peter J. Stuckey</i>	
Estimating Set Intersection using Small Samples .....	71
<i>Henning Koehler</i>	
A Batch Algorithm for Maintaining a Topological Order .....	79
<i>David Pearce and Paul Kelly</i>	
GeoWeight: Internet Host Geolocation Based on a Probability Model for Latency Measurements ....	89
<i>Mohammed Jubaer Arif, Shanika Karunasekera and Santosh Kulkarni</i>	
Joined Q-ary Tree Anti-Collision for Massive Tag Movement Distribution .....	99
<i>Prapassara Pupunwiwat and Bela Stantic</i>	
Structures in Collaborative Tagging: An Empirical Analysis .....	109
<i>Amin Fani Marvasti and David Skillicorn</i>	

Extended Boolean Retrieval for Systematic Biomedical Reviews .....	117
<i>Stefan Pohl, Justin Zobel and Alistair Moffat</i>	
Average Distance as a Predictor of Synchronisability in Networks of Coupled Oscillators .....	127
<i>Anthony Dekker</i>	
Applying a Neural Network to Recover Missed RFID Readings .....	133
<i>Peter Darcy, Bela Stantic and Abdul Sattar</i>	
Analysis of the Periodical Payment Framework using Restricted Proxy Certificates .....	143
<i>Grigori Goldman and Lawrie Brown</i>	
Automated Functionality Testing through GUIs.....	153
<i>Duc Hoai Nguyen, Paul Strooper and Jorn Guy Suess</i>	
Improving End-User GUI Customization with Transclusion .....	163
<i>Lung-Chen Lee, Christof Lutteroth and Gerald Weber</i>	
<b>Author Index</b> .....	173

## 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-Third ACSC in Brisbane, Australia. ACSC 2010 is part of the Australasian Computer Science Week which ran from January 18 to January 22, 2010.

The ACSC 2010 call for papers solicited contributions in all areas of research in Computer Science. This year's conference received 60 submissions from Australia, New Zealand, Japan, Canada, Iran, Iraq, United Kingdom, Finland, Austria, Indonesia and Thailand. The topics addressed by the submitted papers illustrate the breadth of the discipline.

The programme committee consisted of 30 highly regarded academics from Australia, New Zealand, China, Japan, Singapore, Greece and USA. All papers were reviewed by at least three programme committee members, and, in some cases, external reviewers. Of the 60 papers submitted, 18 were selected for presentation at the conference.

Two papers were clear candidates for the "Best Paper Award". As both were also results of higher degree research progress, they are also eligible for the "Best Student Paper Award". Hence, we, as ACSC 2010 Programme Chairs, have been delighted to recommend that they share both prizes.

The "Best Paper Award" was awarded to Mohammed Jubaer Arif, Shanika Karunasekera and Santosh Kulkarni for their paper titled "GeoWeight: Internet Host Geolocation Based on a Probability Model for Latency Measurements", and Matthew Brecknell and Paul Roe for their paper "Modular Transactional Memory".

The "Best Student Paper Award" was awarded to Mohammed Jubaer Arif and Matthew Brecknell for their respective contribution reflected in the two papers.

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 Simeon Simoff (UWS) for his assistance in the production of the proceedings. We thank Professor Justin Zobel for his support as the President of CORE (Computing Research and Education Association of Australasia).

Last, but not least, we express our gratitude to our hosts in Brisbane (in particular Prof Mark Looi and Dr. Wayne Kelly).

**Bernard Mans**

Macquarie University

**Mark Reynolds**

The University of Western Australia

ACSC 2010 Programme Chairs

January 2010

# Programme Committee

## Chairs

Bernard Mans, Macquarie University (Australia) Mark Reynolds, The University of Western Australia

## Members

Steve Blackburn, Australian National University (Australia)  
Stephane Bressan, National University of Singapore (Singapore)  
Fred Brown, The University of Adelaide (Australia)  
Rajkumar Buyya, University of Melbourne (Australia)  
Andy Cockburn, University of Canterbury (New Zealand)  
Curtis Dyreson, Utah State University (USA)  
Vladimir Estivill-Castro, Griffith University (Australia)  
Ansgar Fehnker, NICTA, Sydney (Australia)  
Ken Hawick, Massey University - Albany (New Zealand)  
Xiangjian He, University of Technology Sydney (Australia)  
Michael Houle, National Institute for Informatics (Japan)  
Paddy Krishnan, Bond University (Australia)  
Ming Li, University of Waterloo (Canada)  
Bruce Litow, James Cook University (Australia)  
Brendan McCane, Otago University (New Zealand)  
Maurice Pagnucco, University of New South Wales (Australia)  
Alex Potanin, Victoria University of Wellington (New Zealand)  
Paul Strooper, University of Queensland (Australia)  
Markus Stumptner, University of South Australia (Australia)  
Xiaoming Sun, Tsinghua University (China)  
Antonis Symvonis, National Technical University of Athens (Greece)  
Andrew Turpin, RMIT (Australia)  
Hua Wang, University of Southern Queensland (Australia)  
Thomas Wolle, NICTA, Sydney (Australia)  
Burkhard Wuensche, University of Auckland (New Zealand)  
Masafumi Yamashita, Kyushu University (Japan)  
Yanchun Zhang, Victoria University (Australia)  
Albert Zomaya, The University of Sydney (Australia)

## Additional Reviewers

Ei Ando	Wolfgang Mayer
Peter Andreae	Suleiman Odat
Timothy Bourke	David Pearce
Decheng Dai	Frans Schalekamp
Gavin Finnie	Falk Scholer
Georg Grossmann	Lambert Spaanenburg
Lindsay Groves	Lili Sun
Yanan Hao	Warren Toomey
Bijit Hore	Ian Welch
Guangyan Huang	Kevin Wong
Thomas Kuehne	Guandong Xu
Yiqun Liu	Chee Shin Yeo
Jiangang Ma	Wei Yu
Petra Malik	Anke van Zuylen
Stuart Marshall	

# Organising Committee

## Co-Chairs

Dr. Wayne Kelly  
Prof. Mark Looi

## Budget and Facilities

Mr. Malcolm Corney

## Catering and Booklet

Dr. Diane Corney

## Sponsorship and Web

Dr. Tony Sahama

## Senior Advisors

Prof. Colin Fidge  
Prof. Kerry Raymond

## Finance and Travel

Ms. Therese Currell  
Ms. Carol Richter

## Registration

Mr. Matt Williams

## DVD and Signage

Mr. Matthew Brecknell

## Satchels and T-shirts

Ms. Donna Teague

# Welcome from the Organising Committee

On behalf of the Australasian Computer Science Week 2010 (ACSW2010) Organising Committee, we welcome you to this year's event hosted by the Queensland University of Technology (QUT). Striving to be a "University for the Real World" our research and teaching has an applied emphasis. QUT is one of the largest producers of IT graduates in Australia with strong linkages with industry. Our courses and research span an extremely wide range of information technology, everything from traditional computer science, software engineering and information systems, to games and interactive entertainment.

We welcome delegates from over 21 countries, including Australia, New Zealand, USA, Finland, Italy, Japan, China, Brazil, Canada, Germany, Pakistan, Sweden, Austria, Bangladesh, Ireland, Norway, South Africa, Taiwan and Thailand. We trust you will enjoy both the experience of the ACSW 2010 event and also get to explore some of our beautiful city of Brisbane. At Brisbane's heart, beautifully restored sandstone buildings provide a delightful backdrop to the city's glass towers. The inner city clusters around the loops of the Brisbane River, connected to leafy, open-skied suburban communities by riverside bikeways. QUT's Garden's Point campus, the venue for ACSW 2010, is on the fringe of the city's botanical gardens and connected by the Goodwill Bridge to the Southbank tourist precinct.

ACSW2009 consists of the following conferences:

- Australasian Computer Science Conference (ACSC) (Chaired by Bernard Mans and Mark Reynolds)
- Australasian Computing Education Conference (ACE) (Chaired by Tony Clear and John Hamer)
- Australasian Database Conference (ADC) (ADC) (Chaired by Heng Tao Shen and Athman Bouguettaya)
- Australasian Information Security Conference (AISC) (Chaired by Colin Boyd and Willy Susilo)
- Australasian User Interface Conference (AUIC) (Chaired by Christof Lutteroth and Paul Calder)
- Australasian Symposium on Parallel and Distributed Computing (AusPDC) (Chaired by Jinjun Chen and Rajiv Ranjan)
- Australasian Workshop on Health Informatics and Knowledge Management (HIKM) (Chaired by Anthony Maeder and David Hansen)
- Computing: The Australasian Theory Symposium (CATS) (Chaired by Taso Viglas and Alex Potanin)
- Asia-Pacific Conference on Conceptual Modelling (APCCM) (Chaired by Sebastian Link and Aditya Ghose)
- Australasian Computing Doctoral Consortium (ACDC) (Chaired by David Pearce and Rachel Cardell-Oliver).

The nature of ACSW requires the co-operation of numerous people. We would like to thank all those who have worked to ensure the success of ACSW2010 including the Organising Committee, the Conference Chairs and Programme Committees, our sponsors, the keynote speakers and the delegates. Special thanks to Justin Zobel from CORE and Alex Potanin (co-chair of ACSW2009) for his extensive advice and assistance. If ACSW2010 is run even half as well as ACSW2009 in Wellington then we will have done well.

**Dr Wayne Kelly and Professor Mark Looi**

Queensland University of Technology

ACSW2010 Co-Chairs

January, 2010

# CORE - Computing Research & Education

CORE welcomes all delegates to ACSW2010 in Brisbane. CORE, the peak body representing academic computer science in Australia and New Zealand, is responsible for the annual ACSW series of meetings, which are a unique opportunity for our community to network and to discuss research and topics of mutual interest. The original component conferences ACSC, ADC, and CATS, which formed the basis of ACSWin the mid 1990s now share the week with seven other events, which build on the diversity of the Australasian computing community.

In 2010, we have again chosen to feature a small number of plenary speakers from across the discipline: Andy Cockburn, Alon Halevy, and Stephen Kisely. I thank them for their contributions to ACSW2010. I also thank the keynote speakers invited to some of the individual conferences. The efforts of the conference chairs and their program committees have led to strong programs in all the conferences again, thanks. And thanks are particularly due to Wayne Kelly and his colleagues for organising what promises to be a strong event.

In Australia, 2009 saw, for the first time in some years, an increase in the number of students choosing to study IT, and a welcome if small number of new academic appointments. Also welcome is the news that university and research funding is set to rise from 2011-12. However, it continues to be the case that per-place funding for computer science students has fallen relative to that of other physical and mathematical sciences, and, while bodies such as the Australian Council of Deans of ICT seek ways to increase student interest in the area, more is needed to ensure the growth of our discipline.

During 2009, CORE continued to work on journal and conference rankings. A key aim is now to maintain the rankings, which are widely used overseas as well as in Australia. Management of the rankings is a challenging process that needs to balance competing special interests as well as addressing the interests of the community as a whole. ACSW2010 includes a forum on rankings to discuss this process. Also in 2009 CORE proposed a standard for the undergraduate Computer Science curriculum, with the intention that it be used for accreditation of degrees in computer science.

CORE's existence is due to the support of the member departments in Australia and New Zealand, and I thank them for their ongoing contributions, in commitment and in financial support. Finally, I am grateful to all those who gave their time to CORE in 2009; in particular, I thank Gill Dobbie, Jenny Edwards, Alan Fekete, Tom Gedeon, Leon Sterling, and the members of the executive and of the curriculum and ranking committees.

**Justin Zobel**

President, CORE  
January, 2010

# 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.

**2011.** Volume 33. Host and Venue - Curtin University of Technology, Perth, WA.

**2010. Volume 32. Host and Venue - Queensland University of Technology, Brisbane, QLD.**

**2009.** Volume 31. Host and Venue - Victoria University, Wellington, New Zealand.

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

**2007.** Volume 29. Host and Venue - University of Ballarat, VIC. First running of HDKM.

**2006.** Volume 28. Host and Venue - University of Tasmania, TAS.

**2005.** Volume 27. Host - University of Newcastle, NSW. APBC held separately from 2005.

**2004.** Volume 26. Host and Venue - University of Otago, Dunedin, New Zealand. First running of APCCM.

**2003.** Volume 25. Hosts - Flinders University, University of Adelaide and University of South Australia. Venue - Adelaide Convention Centre, Adelaide, SA. First running of APBC. Incorporation of ACE. ACSAC held separately from 2003.

**2002.** Volume 24. Host and Venue - Monash University, Melbourne, VIC.

**2001.** Volume 23. Hosts - Bond University and Griffith University (Gold Coast). Venue - Gold Coast, QLD.

**2000.** Volume 22. Hosts - Australian National University and University of Canberra. Venue - ANU, Canberra, ACT. First running of AUC.

**1999.** Volume 21. Host and Venue - University of Auckland, New Zealand.

**1998.** Volume 20. Hosts - University of Western Australia, Murdoch University, Edith Cowan University and Curtin University. Venue - Perth, WA.

**1997.** Volume 19. Hosts - Macquarie University and University of Technology, Sydney. Venue - Sydney, NSW. ADC held with DASFAA (rather than ACSW) in 1997.

**1996.** Volume 18. Host - University of Melbourne and RMIT University. Venue - Melbourne, Australia. CATS joins ACSW.

**1995.** Volume 17. Hosts - Flinders University, University of Adelaide and University of South Australia. Venue - Glenelg, SA.

**1994.** Volume 16. Host and Venue - University of Canterbury, Christchurch, New Zealand. CATS run for the first time separately in Sydney.

**1993.** Volume 15. Hosts - Griffith University and Queensland University of Technology. Venue - Nathan, QLD.

**1992.** Volume 14. Host and Venue - University of Tasmania, TAS. (ADC held separately at La Trobe University).

**1991.** Volume 13. Host and Venue - University of New South Wales, NSW.

**1990.** Volume 12. Host and Venue - Monash University, Melbourne, VIC. Joined by Database and Information Systems Conference which in 1992 became ADC (which stayed with ACSW) and ACIS (which now operates independently).

**1989.** Volume 11. Host and Venue - University of Wollongong, NSW.

**1988.** Volume 10. Host and Venue - University of Queensland, QLD.

**1987.** Volume 9. Host and Venue - Deakin University, VIC.

**1986.** Volume 8. Host and Venue - Australian National University, Canberra, ACT.

**1985.** Volume 7. Hosts - University of Melbourne and Monash University. Venue - Melbourne, VIC.

**1984.** Volume 6. Host and Venue - University of Adelaide, SA.

**1983.** Volume 5. Host and Venue - University of Sydney, NSW.

**1982.** Volume 4. Host and Venue - University of Western Australia, WA.

**1981.** Volume 3. Host and Venue - University of Queensland, QLD.

**1980.** Volume 2. Host and Venue - Australian National University, Canberra, ACT.

**1979.** Volume 1. Host and Venue - University of Tasmania, TAS.

**1978.** Volume 0. Host and Venue - University of New South Wales, NSW.



## Conference Acronyms

<b>ACDC</b>	Australasian Computing Doctoral Consortium
<b>ACE</b>	Australasian Computer Education Conference
<b>ACSC</b>	Australasian Computer Science Conference
<b>ACSW</b>	Australasian Computer Science Week
<b>ADC</b>	Australasian Database Conference
<b>AISC</b>	Australasian Information Security Conference
<b>AUIC</b>	Australasian User Interface Conference
<b>APCCM</b>	Asia-Pacific Conference on Conceptual Modelling
<b>AusPDC</b>	Australasian Symposium on Parallel and Distributed Computing (replaces AusGrid)
<b>CATS</b>	Computing: Australasian Theory Symposium
<b>HIKM</b>	Australasian Workshop on Health Informatics and Knowledge Management

Note that various name changes have occurred, which have been indicated in the Conference Acronyms sections in respective CRPIT volumes.

## ACSW and ACSC 2010 Sponsors

We wish to thank the following sponsors for their contribution towards this conference.



CORE - Computing Research and Education,  
[www.core.edu.au](http://www.core.edu.au)



CEED,  
[www.corptech.com.au](http://www.corptech.com.au)



Queensland University of Technology,  
[www.qut.edu.au](http://www.qut.edu.au)



CSIRO ICT Centre,  
[www.csiro.au/org/ict.html](http://www.csiro.au/org/ict.html)



SAP Research,  
[www.sap.com/about/company/research](http://www.sap.com/about/company/research)



AUSTRALIAN  
COMPUTER  
SOCIETY  
Australian Computer Society,  
[www.acs.org.au](http://www.acs.org.au)



Department of Computing,  
[www.comp.mq.edu.au](http://www.comp.mq.edu.au)

# CONTRIBUTED PAPERS



# An Empirical Study of Overriding in Open Source Java

Ewan Tempero

Steve Counsell

James Noble

Department of Computer  
Science  
University of Auckland  
Auckland, New Zealand  
e.tempero@cs.auckland.ac.nz

Department of Information  
Systems and Computing  
Brunel University  
Uxbridge, United Kingdom  
steve.counsell@brunel.ac.uk

School of Engineering and  
Computer Science  
Victoria University of  
Wellington  
Wellington, New Zealand  
kjj@ecs.vuw.ac.nz

## Abstract

Inheritance is a key feature of object-oriented programming. Overriding is one of the most important parts of inheritance, allowing a subclass to replace methods implemented in its superclass. Unfortunately, the way programmers use overriding in practise is not well understood.

We present the first large-scale empirical study of overriding. We describe a suite of metrics that measure overriding and present a corpus analysis that uses those metrics to analyse 100 open-source applications, containing over 100,000 separate classes and interfaces. We found substantial overriding: most subclasses override at least one method and many classes that only declare overriding methods. We also found questionable uses of overriding, such as removing superclass method implementations by overriding them with empty method bodies.

**Keywords:** Overriding, Inheritance, Object-oriented design

## 1 Introduction

Inheritance is one of the defining features of object-orientation and there are many learned articles, books, blogs, courses, and commentaries explaining what it is, and implicitly or explicitly advocating its use. Inheritance is also a source of confusion and unhappiness however, leading Holub to write an article with the title “Why extends is evil” (Holub 2003) and the Gang of Four to exhort us to “Favor object composition over class inheritance” (Gamma et al. 1994). Certainly inheritance is a multifaceted tool: supporting code reuse via subclassing; polymorphism via subtyping; domain modelling via concept hierarchies; and allowing methods to have multiple definitions via overriding. Software developers would like to know when and how inheritance should be used.

We recently conducted a study into the use of inheritance for subclassing and subtyping in Java programs (Tempero et al. 2008). This study found quite high uses of various forms of inheritance: three out of four types were defined using inheritance (not involving Object). That study considered only inheritance between types: it did not examine how those relationships were used in detail, and in particular did

not consider how inheritance between classes affects the methods those classes declare.

While some of the criticisms regarding inheritance might be in fact regarding the use of overriding, this is not to say that overriding should be avoided altogether, and indeed one of the positive examples of use of inheritance is due to the fact that overriding is possible. This means that we need to establish how overriding is actually used, and identify those uses that are questionable or risky. In order to do this, we need some means to characterise how overriding is used. This paper represents a first attempt to answer these questions. Specifically, its contributions are:

1. Present metrics that indicate how overriding is used in a system.
2. Describe, in terms of measurements from the metrics, how overriding is used in 100 open-source Java systems, providing a benchmark for future discussions on the use of overriding.
3. Identify any questionable uses of overriding.

The rest of the paper is organised as follows. In Section 2 we present our motivation in more detail and discuss related work. Section 3 presents details of what metrics we used in our study and what we measured. The results of our study are presented in Section 4, and we give our interpretation of these results in Section 5. Finally we give our conclusions and discuss future work in Section 6.

## 2 Background

### 2.1 Motivation

The “anti-inheritance” arguments refer to so-called “implementation inheritance”, that is, when one class (in Java) **extends** another class and so allows the possibility of some of the implementation of that class to actually be provided by one of its ancestors. When examining the arguments against use of inheritance, the ability for the inheriting class to *override* what it inherits is clearly an important source of concern.

Gamma, commenting on “Favor object composition over class inheritance”, says “But we know that [inheritance is] brittle, because the subclass can easily make assumptions about the context in which a method it overrides is getting called” (Venners 2005). Some of Holub’s examples showing the problems with inheritance rely on the fact that overriding takes place (Holub 2003). One of the issues addressed by Steyaert et al.’s reuse contracts is to deal with situations involving overriding (Steyaert et al. 1996). Overriding is also a feature of the fragile base class problem

(Mikhajlov & Sekerinski 1998). The presence of overriding may not be the only source of problems, but it does seem to be a frequent concern, and so worth studying.

## 2.2 Related Work

Lorenz and Kidd (Lorenz & Kidd 1994) proposed “number of methods overridden” (NMO) as a metric in 1994, however only a few, small, studies using it have been reported.

Briand et al. carried out a study to investigate the relationship between measurements from different metrics and the probability of fault detection in classes during testing (Briand et al. 2000). They looked at 8 C++ systems developed by students, a total of 113 non-library classes. NMO was one of the metrics they investigated. Their results indicated that fault-proneness was correlated to NMO. Of interest to us is to what degree overriding was present in the systems they examined. The largest NMO measurement they observed was 10.

El Emam used the NMO metric as part of an analysis of the effect that class size had on a number of object-oriented metrics (Emam et al. 2001). However due to the fact that they only had 6 non-zero measurements these were excluded from their analysis.

There have been a number of studies using other object-oriented metrics, including depth in inheritance tree (DIT). These have not been directly investigating overriding, but the following are of interest.

An experiment by Daly et al., where the subjects were timed modifying C++ systems with varying levels of inheritance (zero, three and five levels), revealed that a system with three levels of inheritance was easier to maintain than the corresponding system with no inheritance; five levels was however, found to take longer to modify than the equivalent system with zero or three levels of inheritance (Daly et al. 1996).

The Daly et al. study was later replicated, with the results suggesting the opposite effect — that inheritance had a positive effect on maintenance time (Cartwright 1998). Harrison et al. also replicated the experiment and found that the system with zero inheritance was easier to modify than the equivalent systems with three or five levels of inheritance (Harrison et al. 2000). That these studies could get inconsistent results suggests that depth of inheritance is not the only variable that needs to be controlled. We suggest the amount of overriding is something that also needs to be accounted for.

Tempero et al. studied inheritance structures of 93 of the 100 applications we examined in our study (Tempero et al. 2008). That study used a wider interpretation of “inheritance” than we, including, for example, classes that implement interfaces. The study did separate out measurements for implementation inheritance, which indicated that implementation inheritance was not at all rare, and in fact could be quite significant in some applications.

Baxter et al. looked at the distributions of a number of metrics for object-oriented code in 56 Java applications (most of which are in this study) (Baxter et al. 2006). In almost all cases the distributions they observed were one of two classes of distributions — powerlaws and so called “truncated curves” that are almost, but not quite, powerlaws (such as log-normal or stretched-exponential). The Tempero et al. study also noted the same two classes of distributions.

## 3 Methodology

As noted in the previous section the metric that has been generally used regarding overriding is NMO, the

number of superclass methods a subclass overrides. In carrying out our study, we quickly found there are many different characteristics of overriding that we wanted to examine, leading us to develop many new metrics. For this particular study, we defined and used more than 25 metrics, and there are more still we could have considered. We do not have space to present all these results, and so rather than listing every potential metric, we motivate the need for multiple metrics here, and then define a practical set of metrics. The next section presents the result of these metrics applied to our Java corpus.

Before discussing metrics, we need to be specific about the *entities* the metrics are to apply to and the *attributes* we wish to measure (Fenton & Pfleeger 1998). NMO is typically defined in terms of “classes” and “methods.” Java, however, has many “class-like” things, such as “interfaces”, and its “methods” may or may not have implementations, so it is not always clear what NMO means. There are several possible decisions we could make. As this is our first study of overriding, we have chosen the simplest set we thought would be useful, as we now describe.

We want to measure how overriding is actually used, and to do so we must be precise about what we mean by “overriding”. For this study, we focus on overriding that results in *replacement of implementation*, as that is a common theme in the arguments against the use of inheritance. This means that we do not consider abstract methods, as there is no implementation being replaced in such cases. There is also no choice regarding abstract methods — they must be “overridden”. The decision to make a method abstract seems to be different in nature to the decision to replace an implementation, and so we prefer to measure those decisions separately (however see the discussion regarding “trivial” overriding in section 4.5). Note that this means our notion of *overriding* is not exactly that from the Java Language Specification (Gosling et al. 2005, 8.4.8).

Thus, for our study, a method *m* in class *B* *overrides* a method from class *A*, an ancestor of *B*, if it overrides in the sense of Java, and in addition, *m* must be implemented in *A* and *m* must be implemented in *B*, that is, for us, overriding takes place when one implementation gets *replaced* by another. The implementation being replaced is *overridden* and the implementation it is replaced by is *overriding*.

Our definition is illustrated in figure 1. In this example the implementation *a1()* in *B* overrides *a1()* inherited from *A*, however *a2()* in *B* does not override anything as no implementation of that method is inherited from *A*. Note that *a1(int)* in *B* is not involved in any overriding, being an example of overloading — *a1()* and *a1(int)* are two different methods with different signatures, although they share the same name.

Since constructors and static and private methods cannot be overridden, they are not considered in our metrics. Synthetic methods are also not considered as they are not written by developers. We do include native methods. It is debatable whether we should include them, however overriding a native method (or vice versa) is still replacing implementation, and we do not expect this to happen so much as to significantly change our results one way or the other.

We want to try to characterise the decision to override a method. The developer has no control over the definition of classes from the standard library or third-party libraries, but does have some control over other user-defined classes, consequently we believe this decision is different. As we do not have space to show both, in this paper we limit the results to just overriding from other user-defined classes.

As we are interested only in implementation, we do not consider interfaces (and annotations). While

```

abstract class A
public void a1(){}
abstract public void a2();
public void a3(){}
}
class B extends A {
public void a1(){}
public void a1(int i) {}
public void a2(){}
public void b1(){}
}

```

Figure 1: The definition of overriding. The method `a1()` in `B` overrides `a1()` inherited from `A`, however `a2()` in `B` does not override anything as no implementation of that method is inherited from `A`. The implementation of `a1()` in `A` is said to be *overridden* and the implementation in `B` is *overriding*.

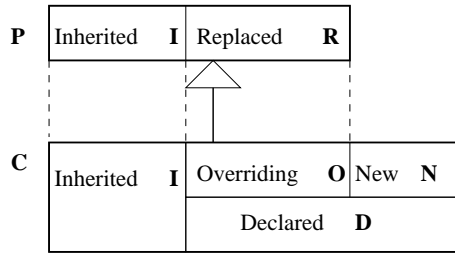


Figure 2: Modelling overriding for individual classes

exceptions are in some ways just like any other class syntactically, they are treated specially by the Java run-time, and have quite a different role from other “normal” classes. We feel that for the moment we will get a clearer picture of how developers use implementation inheritance if we ignore exceptions. For similar reasons, we count inner classes, but we ignore generic parameters, as in Java they do not affect method overriding. Note that we do measure abstract classes, although if none of their methods are implemented (as happens) then they will have no measurement in that case.

As we do not consider classes from the standard library or third-party libraries, an issue arises in that for any user-defined class that extends such classes, we cannot determine what is actually being inherited. For this reason, we only consider methods inherited from other user-defined classes, that is, if (for example) a class overrides a method from the standard library, that fact will not be represented in our measurements.

To summarise: we measure only user-defined classes, including abstract classes and all types of inner classes; and only non-private (public, package and protected) non-native methods within those classes; and we treat generic classes according to their erasure.

### 3.1 Metrics overview

Our **method** metrics are based on figure 2. The figure shows two classes related by inheritance, where the lower or *child* class `C` inherits from the upper or *parent* class `P`. We are only interested in the non-private instance methods of a class, which we collectively refer to as its *protocol*. The protocol methods of the two classes can be categorised as follows. `P`’s methods are either **inherited**, meaning their implementations in `C` is the same as in `P`, or their implementations are overridden or **replaced** in `C`. The methods of `C` can be categorised as **inherited**, meaning their

Methods in classes		
Super	<b>I</b> <b>R</b> <b>NMIC</b>	implementation is Inherited implementation is Replaced $ I  +  R $
Sub	<b>I</b> <b>O</b> <b>N</b> <b>D</b> <b>NMC</b>	(as above) implementation is Overriding implementation is New implementation is Declared $ D  +  I $
Classes in applications		
Super	<b>CR</b> <b>CN</b>	Classes that have implementations Replaced Classes with No implementations replaced
Sub	<b>CO</b> <b>CI</b> <b>NCI</b>	Classes that Override at least one method Classes that only Inherit $ CO  +  CI $
Methods in applications		
Super	<b>MR</b> <b>MN</b>	Methods that have implementations Replaced Methods with No implementations replaced
Sub	<b>MO</b> <b>MI</b> <b>NMI</b>	Methods that are Overriding Methods that are Inherited $ MR  +  MN $

Figure 3: Notation guide to metrics for measuring overriding.

implementations come from `P`, or **declared**, meaning their implementations are defined in `C`’s definition. The **declared** methods can be further divided into **overriding**, meaning their implementations replace those in `P`, or **new**.

To distinguish metrics related to parent and children classes, we use the term “replaced” for parent methods, rather than “overridden”; we use the term “overriding” only for child methods. For any given pair of parent and child classes the number of replaced methods must be the same as the number of overriding methods — the alignment in the figure is intended to indicate this. For a given parent class, the sets **I** and **R** may differ for different child classes.

Several metrics relating to overriding can be derived from figure 2, such as sizes of the various sets (e.g. the number of overriding methods  $|O|$ ), or proportions of one set with respect to another (e.g., the proportion of overriding methods to those declared  $|O|/|D|$  or the proportion of methods replaced  $|R|/(|I| + |R|)$ ). These metrics have classes as their entities (that is, a measurement is for a single class).

We generalise figure 2 to define metrics over all classes and all methods in applications (that is, taking whole applications as entities and measuring classes and methods). For the **classes in applications** metrics, the top (super) box contains all classes that have subclasses and the bottom (sub) box has all subclasses. Classes are then categorised as to whether or not they contain methods that have implementations replaced or that override methods in their parent class. For the **methods in applications** metrics, the super box contains all methods belonging to classes that have subclasses, and the sub box contains all methods belonging to subclasses. The methods are categorised according to whether or not their implementation is replaced or they override another implementation.

Figure 3 summarises these metrics, and gives the notation we use. Finally, we use **NC** for the number of classes (not interfaces, enums, annotations, or exceptions) that have been developed for the applica-

tion, counting nested classes as individual classes.

### 3.2 Population Measured

We base our empirical study on the Qualitas Corpus (Qualitas Research Group 2008) a collection of open source Java applications we have been using for similar studies. The version we used was released on 3 June 2008 and consists of 100 distinct applications, including multiple releases of some applications. The details, including the full list of applications studied, is available on the Qualitas Corpus website. The measurements were performed on the bytecode representation of the applications. While this is not a true representation of the source code, all information regarding class relationships and method declarations is available from bytecode, so this is sufficient for our study.

## 4 Results

### 4.1 Application Measurements

We begin by looking at measurements for applications. Figure 4 shows the number of classes that override something ( $|\text{CO}|$ ) along with the number of classes that inherit something ( $\text{NCI}$ ) and the number of classes in the application ( $\text{NC}$ ). The chart has been truncated in the y-axis to make the scale clearer. It shows the distribution of the population we studied. There are 24 applications with more than 1000 classes and 5 with more than 2000 classes (these shown in table 1).  $\text{NCI}$  and  $|\text{CO}|$  both roughly (but not exactly) follow  $\text{NC}$  in that the larger applications tend to inherit and override more.

Figure 5 shows the number of classes that inherit from other user-defined classes as a proportion of user-defined classes (that is  $\text{NCI}/\text{NC}$ ). This is the UDCCUI metric used by Tempero et al (Tempero et al. 2008). As we will do in other cases, the x-axis is the *rank* of an application for a given size metric, rather than its size. There are many applications with similar sizes, meaning plotting against size would make the individual measurements difficult to distinguish. The applications are ordered according to the appropriate size metric ( $\text{NC}$  in this case) in order to see if there is a trend with respect to size.

Figure 5 is useful to help evaluate the later results, but it is also useful to see the degree to which developers have tried to inherit implementation. Of the 100 applications, 31 have at least half of their classes inheriting from other user-defined classes (recall that this does not include classes that inherit from standard library or third-party classes). The minimum is 4% (*mvnforum*), the maximum is 85% (*jparse*), and the median is 38% (*drawswf*). There is no obvious trend with respect to size; the right-most 24 (more than 1000 classes) measurements are not at all different to the left-most 76.

### 4.2 Classes that Override

Figure 6 shows the proportion of classes that have at least one method that overrides another method to the number of classes that could possibly have such methods, that is, the number of classes that inherit from some other user-defined class. While there are some applications that have low proportions, they are applications that do not have many classes that inherit from other classes. The minimum is 0% (*jasm1*,  $\text{NCI}=21$ ), median is 53% (*trove*,  $\text{NCI}=125$ ), and maximum is 100% (*nekohtml*,  $\text{NCI}=6$ ), which also has the smallest number of user-defined classes. The application with the largest number of classes is

Application	CO	NCI	NC
gt2	900	1436	2515
jtopen	374	931	2605
azureus	409	1261	4159
jre	3240	5828	11181
eclipse	6042	8983	17621

Table 1: The five largest applications studied, as measured by  $\text{NC}$ .

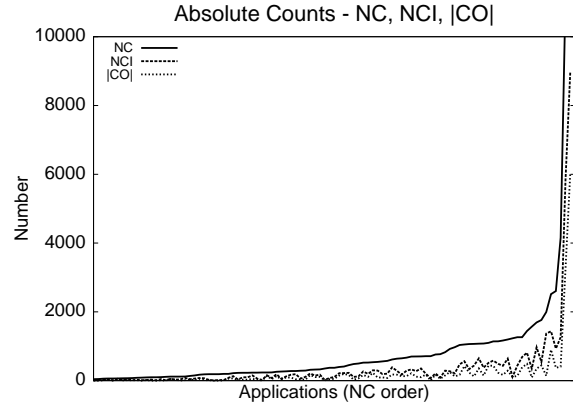


Figure 4: Number of user-defined classes ( $\text{NC}$ ), Number of classes that inherit ( $\text{NCI}$ ) and Number of classes that override ( $|\text{CO}|$ ). (Truncated in y-axis — see Table 1.)

*eclipse* (17621), which is at 67%. The next largest is *jre* (11181) with 56% of its classes that inherit overriding at least one inherited method. The application with the second-largest proportion is *ireport*, which has 112 classes that inherit, but its proportion of classes that inherit is relatively small (112/1232, 8%). Again there is no obvious trend with respect to size. We also looked at whether the degree to which inheritance is used is related to how much overriding takes place and found no obvious patterns.

We can also consider how often, on average, a class overrides methods. We start with a simple average over all classes in an application. Figure 7 shows two averages — one is the average number of methods overridden over all classes that inherit something ( $|\text{MR}|/\text{NCI}$ , circle) while the other shows the average number of methods overridden over those classes that override something ( $|\text{MR}|/|\text{CO}|$ , squares). There is one application (*nekohtml*) where the two averages are the same, indicating that every

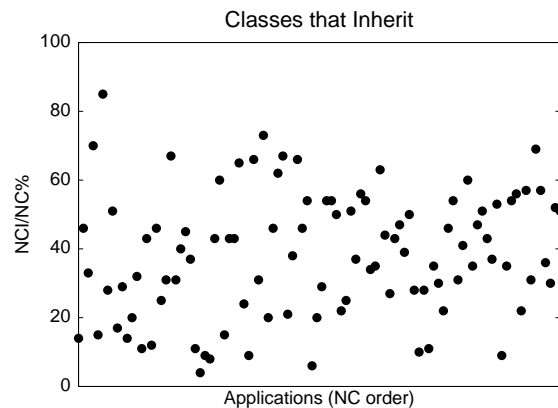


Figure 5: Proportion of Classes that Inherit of user-defined Classes ( $\text{NCI}/\text{NC}$ ) for each application in order of Number of user-defined Classes ( $\text{NC}$ ).



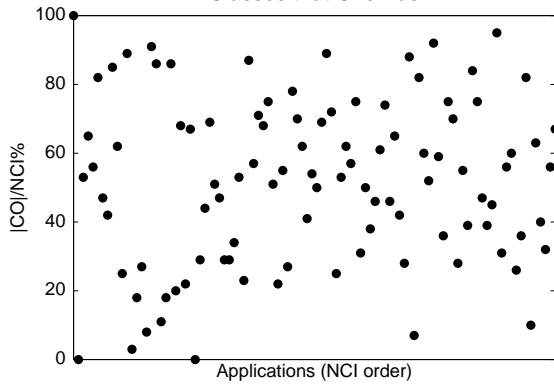


Figure 6: Proportion of Classes that Override of Classes that Inherit ( $|CO|/NCI$ ) for each application in order of Number of Classes that Inherit ( $NCI$ ).

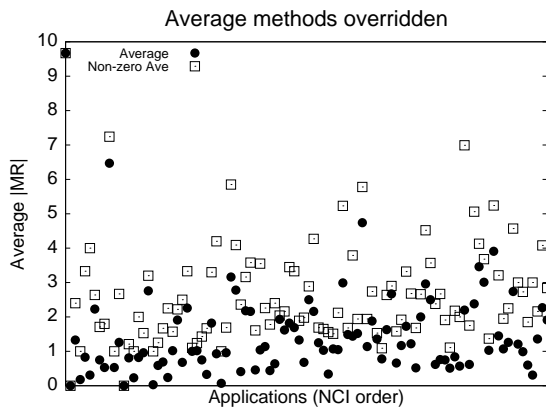


Figure 7: Circles show average Number of Methods that have implementations Replaced ( $|MR|$ ) for each application in order of Number of Classes that Inherit ( $NCI$ ). Squares show average  $|MR|$  considering only classes that override ( $CO$ ).

class that inherits overrides something. This application also has the largest average (9.67), however there are only 6 classes that inherit anything. The median  $|MR|/NCI$  is 1.14, 42 applications have an average of 1 or less, 77 applications have an average of 2 or less, and 93 have an average of 3 or less. The largest application, *eclipse*, has an overall average of 1.91, that is, for every class in *eclipse* that inherits from another class in *eclipse*, on average nearly two of its methods are overriding.

While there are a number of applications with a low degree of overriding, there are also those where it is quite high. This suggests that the way overriding is used might be due to either the domain or the programming style (also see Section 4.4). For any subclass in an application, there is likely to be at least one method that overrides an inherited method.

### 4.3 Methods that are Overridden

In the results we have presented so far we have considered a class to “use overriding” if it has *one* method that overrides another. It may be that the same method is overridden many times (when its class has many descendants). To get an overall view of this, Figure 8 shows the number of inherited methods in an application that are overridden by something else as a proportion of the total number of inherited methods ( $|MR|/NMI$ ).

While there are some high values (the highest is 82% — *nekohtml* again), the median is 18%

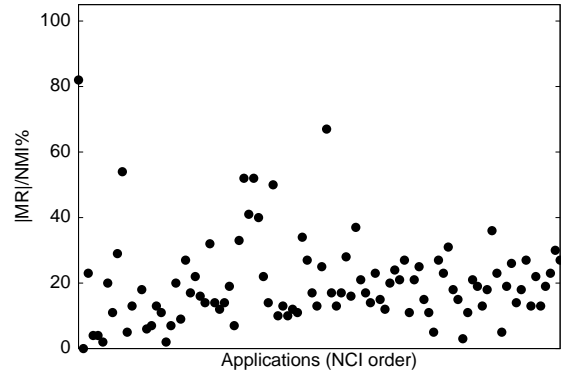


Figure 8: Proportion of Number of Methods Replaced of Number of Methods that are Inherited ( $|MR|/NMI$ ) for each application in order of Number of Classes that Inherit ( $NCI$ ).

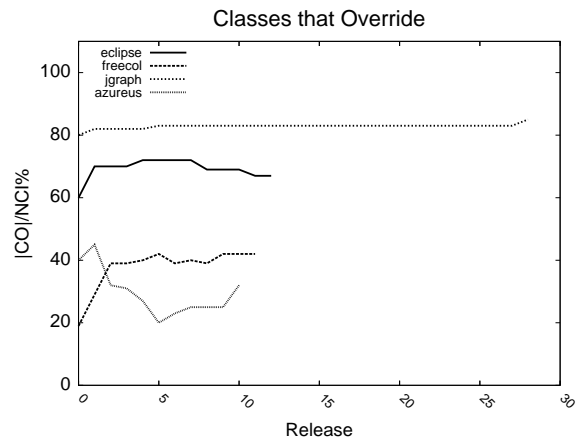


Figure 9: Proportion of classes that override to class that inherit ( $|CO|/NCI$ ) for 4 applications. (Note the  $x$ -axis is merely a release number, and so it not equivalent for all applications.)

(*hibernate* — 509 methods overridden out of 2822 inherited), the smallest is, unsurprisingly, *jasml* with 0%. The application that overrides anything with smallest value is *informa* at 2% (4 out of 171 methods inherited) and overall 75% of the applications have 25% or fewer methods that get overridden.

### 4.4 Longitudinal Results

Another view of the data is a longitudinal one. We tracked the measurements of each of the releases we have for *jgraph* (29 releases), *freecol* (12), *azureus* (11), and *eclipse* (13). Figure 9 shows the results for the proportion of classes that have a method that overrides to the classes that inherit ( $|CO|/NCI$ ). In interpreting this figure, it must be noted that the releases of *jgraph* come from a shorter period of its entire lifespan than the other applications, as the earliest release we have is relatively late in its development compared to the other applications, which would explain the fairly straight line. While *jgraph* is both the smallest of the 4 and has the highest measurements, the largest application, *eclipse*, has the next highest measurements. The other two applications go through significant changes.

Figure 10 shows the proportion of the overridden methods to the total number of methods inherited ( $|MR|/NMI$ ). Here we see quite similar curves, although not close enough to be a common trend. The relatively flat nature of all curves indicates that as applications grow in size (as they all mainly did over

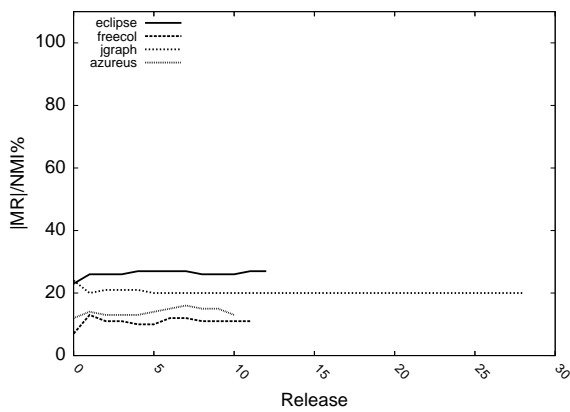


Figure 10: Proportion of methods that are overridden to methods that are inherited ( $|MR|/NMI$ ) for 4 applications. (Note the  $x$ -axis is merely a release number, and so it not equivalent for all applications.)

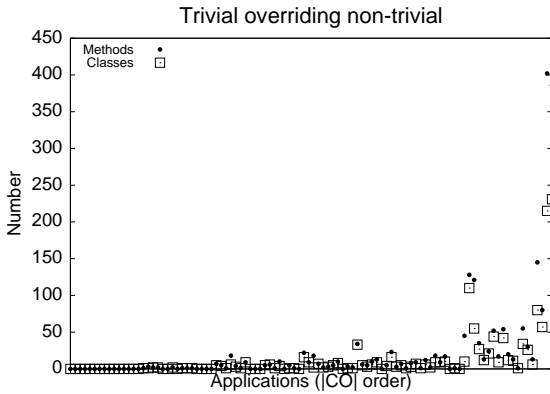


Figure 11: Degree of trivial overriding.

the releases we studied) the number of overridden methods also grows at the same rate, although figure 9 suggests the distribution of such methods to classes changes. This is further evidence that the use of overriding is related to either domain or programming style. Determining which is the case would be an important result.

#### 4.5 Presence of Trivial Implementations

When manually examining code, we noticed a number of cases of overriding where the bodies of the methods were either empty, or returned a constant, which we call “trivial” implementations. For example, the abstract class `org.apache.tools.ant.Task` has methods that have empty bodies (such as `init` and `execute`) that are overridden by inherited classes. While it might make sense that `init` has a default implementation that does nothing, it is not clear why `execute` should, and in fact `execute` is overridden by every subclass. Since `Task` is abstract, there is no obvious reason why `execute` could not be made abstract. Classes with empty implementations that are overridden may or may not make sense, but identifying such cases may provide opportunities for changing design choices (or at least improving documentation).

There are also cases where both the overridden and the overriding methods are empty. For example the `init` method in `Task` is overridden by an empty implementation in `org.apache.tools.ant.taskdefs.optional.sound.SoundTask`. While this seems harmless (especially in this case) it is still a questionable situation that deserves clarification.

More troubling is when a non-trivial method has

been overridden by an empty method. For example, `org.apache.tools.ant.DirectoryScanner` has methods (e.g. `AddDefaultExcludes`) that have non-trivial implementations that are overridden in `org.apache.tools.ant.types.optional.depend.DependencyScanner` by empty implementations. This throws away inherited implementation, which seems very questionable. In principle, this situation can be removed by having a common (possibly abstract) superclass to `DirectoryScanner` and `DependencyScanner` containing the shared implementation and then each class providing their specific (possibly trivial) implementation. Identifying such situations indicate possibilities for refactoring.

We measured how many methods there were in an application that had empty bodies but overrode methods with non-empty bodies, and how many classes had such methods. The results are shown in figure 11, where the number of classes are shown as boxes, and the number of methods as circles. As can be seen, for most applications the numbers were fairly small, with 31 applications having no such methods.

The figure also suggests that there is often only one such method per class, which is the case for 57 applications. The maximum number of classes is in `eclipse` (231, 386 methods overall) whereas the maximum number of methods is `jre` (402 in 215 classes). The degree to which such methods exist is not completely related to the amount of overriding with `compiere` (424 classes with overriding, 8th most) having just 1, and `weka` (445 classes with overriding, 5th largest) having 6.

Our conclusion is that there is a significant degree of this questionable use of overriding. These results suggest that it is worthwhile providing IDE support to identify when this kind of use occurs and more refactoring support to help remove it.

#### 4.6 Class Measurements

We now turn to the measurements for individual classes. Here our main interest is in the shapes of the distributions, rather than individual values. As we cannot show all distributions of all applications, we have picked two representative applications. Figures 12 and 13 show results for `jgroups` and `eclipse` side-by-side. `Eclipse` is the largest application we studied ( $NC=17621$ ) whereas `jgroups` is relatively small (924, although it is the 75th largest in the corpus by  $NC$ ). Points to note are:

- The first column of figure 12 shows the Number of methods Declared ( $|D|$ ). This is a almost-but-not-quite-power-law (truncated curve) as observed previously (Baxter et al. 2006).
- The next column shows Number of methods Overridden ( $|O|$ ), again showing truncated curves.
- The third column shows the Number of Methods Inherited ( $NMIC$ ).
- The last column shows Number of Methods ( $NMC$ ). Recall that this metric considers only non-private, implemented methods in a class, but includes both those declared in the class and those inherited (but not overridden).
- In figure 13, the top row shows how many classes have a given proportion of methods that override to the methods that are inherited ( $|O|/NMIC$ ). As Figure 7 indicates, for most applications, when a class overrides something, it typically overrides only 1 method. Yet the two charts shown are by no means unique. Of particular

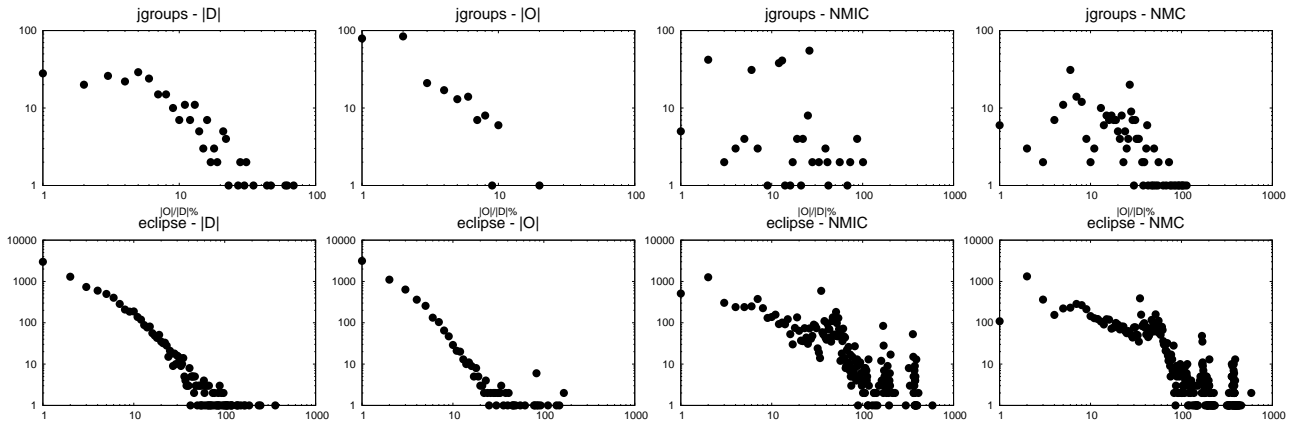


Figure 12: Log-log frequency distributions of measurements for **jgroups** (top) and **eclipse** (bottom). X-axis left to right: Number of Methods Declared in a Class ( $|D|$ ), Number of Methods Overridden in a Class ( $|O|$ ), Number of Methods Inherited in a Class (NMIC), Number of implemented Methods in a Class (NMC).

interest is the number of classes in **jgroups** that override all of the methods inherited (38 of 273 classes that inherit something, or 14%). This is quite different to **eclipse**, but again not unique.

- The bottom row of Figure 13 shows how many classes have a given proportion of methods that override of the methods that are declared ( $|O|/|D|$ ). Of note here is the number of classes in both applications where all of the methods declared in a class are overriding an inherited method although the proportion of classes in this category are quite different (for **jgroups**, it is 39 out of 273, or 14%, and for **eclipse**, it is 2955 out of 8983, or 33%).

We now focus on the classes that either override all inherited methods (Figure 14) or declare only methods that override inherited methods (Figure 15).

Figure 14 shows the number of classes that override all inherited methods as a proportion of the number of classes that inherit something. The application with the largest proportion of such classes is **galleon** (19%). The application with the largest absolute number of such classes is in fact the second-largest application, **jre** (176 classes that override all inherited methods, where  $NCI=5828$ , or 3%).

Classes that override all inherited methods get little benefit from implementation inheritance. It is possible that in some cases they benefit through the use of **super** but such large numbers might be due to the fact that there are many classes that don't inherit many methods (e.g. only 1 method is inherited) but this is an area that deserves further study.

We looked at some of the classes that overrode everything that was inherited, concentrating on those classes that inherited 10 or more methods. There were 9 such applications with **jre** having 11 classes, **poi** 4, **eclipse** 2, and **javacc**, **jspwiki**, **c\_jdbc**, **drjava**, **jrubby**, and **gt2** one each.

- **org.javacc.jjdoc.HTMLGenerator** 18 methods inherited from **org.javacc.jjdoc.Generator**. This is the only class that inherits from its parent, which provides text generation. These two classes could easily be redesigned to implement a common interface.
- **com.ecyrd.jspwiki.TranslatorReader\$HTMLRenderer** 36 methods inherited from **com.ecyrd.jspwiki.TranslatorReader\$TextRenderer**. Comments in the source indicate that further work is

intended with respect to the design of this class. Again this class and the class that it inherits from could implement a common interface.

- **javax.swing.plaf.synth.ImagePainter**, **javax.swing.plaf.synth.ParsedSynthStyle\$DelegatingPainter** 113 methods inherited from **javax.swing.plaf.synth.SynthPainter**, an abstract class. In this case these aren't the only subclasses of **SynthPainter**. **com.sun.java.swing.plaf.gtk.GTKPainter** also inherits. But again a common interface seems possible.

This sample suggests that those classes that override everything they inherit, and the other classes that participate in the same inheritance hierarchies, are candidates for refactoring by introducing appropriate interfaces.

Figure 15 shows the number of classes that declare only methods that override something. The largest is **freecs** with 87% and the median is **jgrapht** (15%). This is clearly a much more common situation.

#### 4.7 Depth of Inheritance

We might expect that classes deeper in the inheritance hierarchy are bigger than those at the top. To investigate these claims we looked at the average size of classes at different levels of inheritance. Figure 16 shows the average values for NMC, NMIC,  $|O|$ , and  $|D|$  for each application for  $DITCCUD=1$  and  $DITCCUD=5$ . We chose  $DITCCUD=5$  because at deeper levels of inheritance we have fewer classes and so trends are not as clear as they are at  $DITCCUD=5$ , although they are consistent with what we show. For comparison, we also show the average  $DITCCUD$  per application for those classes that inherit from user-defined classes (and so the  $DITCCUD$  is at least 1).

The first thing to note is that the average depth of inheritance trees seems largely unrelated to the size of the application (as measured by  $NCI$ ). While the maximum depths increase with application size, the number of shallow classes also increases with size.

The NMC measurements confirm the expectation that, on average at least, the number of implemented methods for a class increases with depth in the inheritance tree. It is interesting that this is more obvious with the larger applications, despite the fact that their average  $DITCCUD$  values are no bigger than the smaller applications.

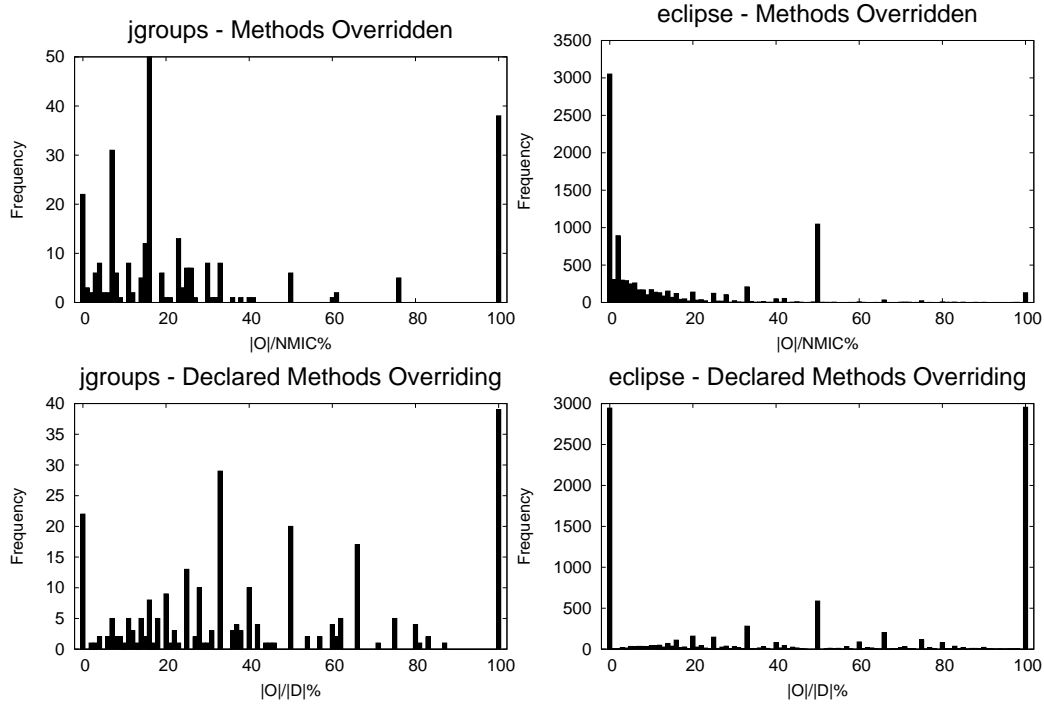


Figure 13: Frequency distributions of measurements for *jgroups* (left) and *eclipse* (right). Proportion of methods overridden to methods inherited ( $|O|/NMIC$  — top) and Proportion of methods overridden to methods declared ( $|O|/|D|$  — bottom).

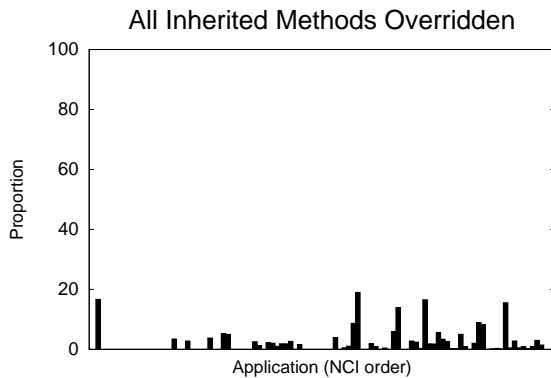


Figure 14: Proportion of classes that inherit of classes that override all inherited methods.

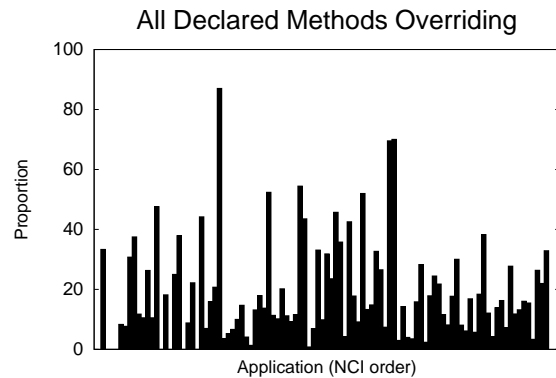


Figure 15: Proportion of classes that inherit with all declared methods overriding something.

The number of methods inherited (NMIC) are also larger at the deeper levels, which is not surprising as if there are more methods at deeper levels then there are more methods to inherit. The number of methods overridden is not so conclusive. There is one large application (*jtopen* NC 2605) that has a high average, but that one value is not enough to indicate a trend. Note that the scale for the  $|O|$  measurements is significantly smaller than for NMC and NMIC.

The number of methods declared ( $|D|$ ) does not appear to be any different at different depths in the inheritance tree. In fact, generally  $|D|$  is smaller at deeper levels, which is what we would expect. The fact that the number of methods declared at deeper levels is anything like the number at the top of the inheritance tree is probably due to the number of overridden methods, which  $|D|$  includes.

## 5 Discussion

For the question “how much overriding is there” we can say this: half the applications have 53% or more of their subclasses using overriding (figure 6), and 58 applications average more than one method overridden per subclass (figure 7). These results are largely independent of the size of the application. In other words, anyone dealing with a subclass has a better than even chance of having to also deal with overriding. Furthermore, this is overriding of methods written for the application. We do not include overriding of third-party or standard library methods, which clearly also takes place. If overriding is a source of difficulty then what we have observed suggests it would contribute a non-trivial cost.

Of course the ability to override does have some benefits, and so this cost may be justified by the benefits. While we have not specifically looked at the benefits accrued by overriding, we noted two prominent characteristics of actual overriding whose benefits are very questionable. These are:

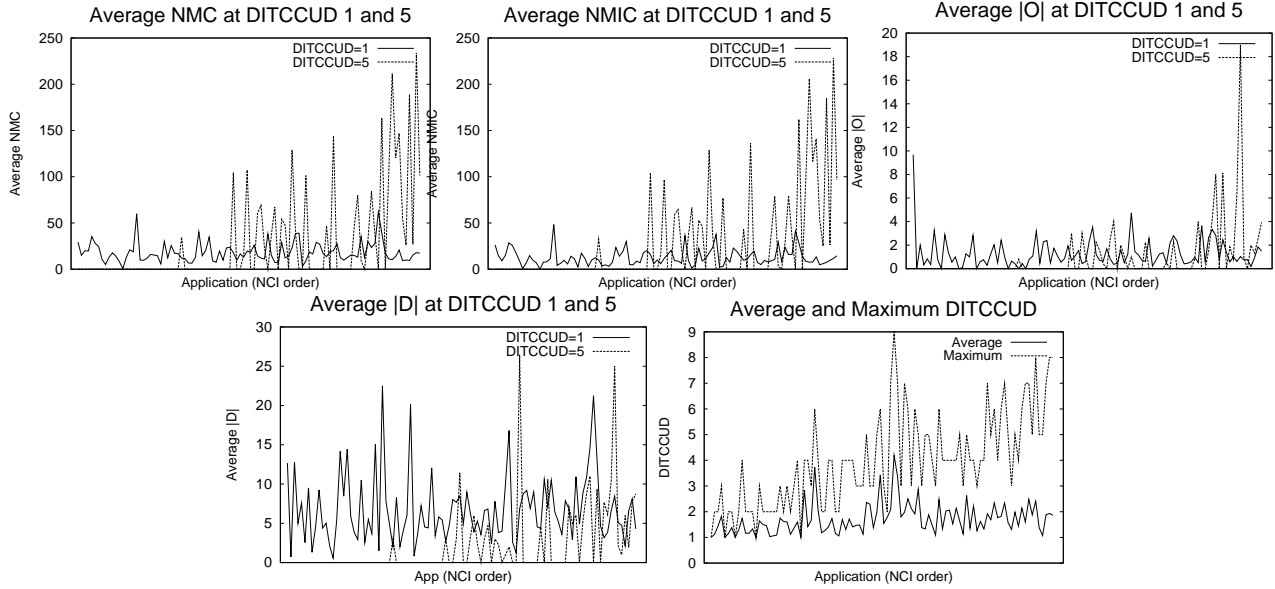


Figure 16: Average measurements for each application in **NCI** order for (left to right, top to bottom) **NMC**, **NMIC**, **|O|**, and **|D|** for classes with **DITCCUD** 1 and 5. The average and maximum **DITCCUD** measurements are shown for each application for comparison. Lines are used to emphasise trends.

- The number of cases where either the overridden or overriding method is trivial. We believe that when this occurs, there is the possibility that the inheritance structure can be refactored using interfaces (or at least abstract classes) to remove this use of overriding. In particular, if the overriding method is trivial (and especially if the overridden method is also trivial) there is significant room for improvement.
- The number of cases where all inherited methods are being overridden. In such cases, there appears to be no benefit due to implementation inheritance, and so refactoring again seems possible to remove the overriding.

Given the results relating number of overriding methods to fault proneness (Briand et al. 2000), it would seem a good first start would be to remove as much of this trivial overriding as possible.

IDEs such as Eclipse already give an indication (typically a decoration in the editor) when overriding occurs. This means the infrastructure exists to be able to signal to developers when these cases have occurred, allowing proactive assessment. Further, we believe that these observations can be exploited to provide automated refactoring support, that is, in some cases it should be possible to automatically determine how to restructure the inheritance hierarchy to remove overriding.

Figure 12 suggests we should treat averages with caution. Whether or not the number of methods overridden in a class (**|O|**) is a powerlaw, it is certainly highly skewed. While most classes that override anything override only a few methods, there are classes that override many methods (up to 100 in the case of **eclipse**). We need to better understand why so many methods need to be overridden. Some cases can be explained by the provision of default implementations for large interfaces. For example, the visitor pattern when used for parsers results in very large interfaces. However this cannot explain all measurements we see.

Another question we need to consider is, should there be more overriding? Is it the case that the advice of people such as Gamma is being followed and, with less use of inheritance, there is less opportunity

to exploit the benefits of overriding? Answering this question requires better characterisation of benefits of overriding than exists, and we feel developing such a characterisation ought to be a priority if we are to use inheritance to its fullest benefit.

One possibility that we have to consider is if alternative mechanisms to inheritance, and hence to overriding, were used instead. In C++, the friend facility was often used to circumvent the need to re-engineer the inheritance hierarchy (even in a relatively small way); empirical studies have shown that friends are indeed used frequently and may also be the cause of possible faults (Counsell & Newson 2000, Briand et al. 1997). Our study did not consider Java interfaces or abstract methods in its analysis. One suggestion could be that the use of interfaces may be the mechanism through which complex inheritance relationships (even multiple inheritance as we know it in C++) are avoided.

## 5.1 Threats to Validity

The main threat to the validity of our conclusions is the corpus we used, which consists entirely of successful open-source Java applications, many of small to medium size. Our results do apply to at least these applications, many of which (e.g. **eclipse**, **ant**) are some of the most used Java programs worldwide, and so we are confident that our conclusions hold within this population. We do not know whether or not the level of overriding is different for applications created under different development models than open source, or in different languages, or in applications that are unsuccessful. More empirical studies are needed to determine this.

## 6 Conclusions

We have presented a large empirical study on the use of overriding in Java open source software, considering 100 applications with over 100,000 user-defined types. We have developed a general approach to investigating overriding, and a set of metrics for measuring different aspects of overriding. We have found substantial use of overriding — every subclass in an

application is likely to override at least one inherited method. We also discovered two common uses of overriding that are questionable, namely when:

- classes override every inherited method, and
- when classes provide empty implementations that override non-empty implementations.

We believe these two situations represent unnecessary overriding, and further, are amenable to automatic detection and support for removal. Our experience is that these kinds of findings are only possible through large-scale empirical studies.

This is by far the largest study of this kind. It provides a benchmark of the level of overriding used in this population that is typical today. For example, our results can be used to identify suitable candidates in the corpus for study on the effects of use of overriding on development or maintenance costs. This benchmark can also be used to evaluate the use of overriding in future studies, both in Java open source software and in other languages supporting software.

There are a number of directions future research could take. We only considered overriding methods from other user-defined classes in this study. We need to also examine how overriding is used with respect to the standard library and third-party libraries. We have begun looking at how `super` is used. Its use could be the source of difficulty with respect to overriding, as indicated by the so-called “Yo Yo” problem (Binder 1999).

Our study needs to be replicated by others to verify our results. Other populations, such as commercially-developed Java applications and applications developed in other languages, need to be studied. We have noted our corpus consists of what we have called “successful” applications. It would be very interesting to see if “unsuccessful” applications (particularly those that have been unsuccessful due to difficulty in maintaining the code) exhibit significantly different levels of overriding.

The benefits overriding provides needs to be better characterised, or at least more indicators of questionable use of overriding determined. For example, we have noted occurrences of overriding where the replacement implementation is identical to what it is replacing (trivial, in both cases). Does it happen that identical non-trivial overriding takes place? It seems unlikely, but we won’t know for sure until we look. It is only through empirical studies of the kind we have presented that such questions can be settled.

## Acknowledgements

Most of this work was done while Tempero was a visiting researcher to the BESQ project at Blekinge Institute of Technology, Ronneby, Sweden, whose support he gratefully acknowledges.

## References

- Baxter, G., Frean, M., Noble, J., Rickerby, M., Smith, H., Visser, M., Melton, H. & Tempero, E. (2006), Understanding the shape of Java software, in W. Cook, ed., ‘OOPSLA’, pp. 397–412.
- Binder, R. V. (1999), *Testing object-oriented systems: models, patterns, and tools*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- Briand, L. C., Wüst, J., Daly, J. W. & Porter, D. V. (2000), ‘Exploring the relationships between design measures and software quality in object-oriented systems’, *Journal of Systems and Software* **51**(3), 254–273.
- Briand, L., Devanbu, P. & Melo, W. (1997), An investigation into coupling measures for C++, in ‘ICSE’, pp. 412–421.
- Cartwright, M. (1998), ‘An empirical view of inheritance’, *Information and Software Technology* **40**, 795–799.
- Counsell, S. & Newson, P. (2000), ‘Use of friends in C++ software: an empirical investigation’, *Journal of Systems and Software* **53**(1), 15–21.
- Daly, J., Brooks, A., Miller, J., Roper, M. & Wood, M. (1996), ‘Evaluating inheritance depth on the maintainability of object-oriented software’, *Empirical Software Engineering* **1**(2), 109–132.
- Emam, K. E., Benlarbi, S., Goel, N. & Rai, S. N. (2001), ‘The confounding effect of class size on the validity of object-oriented metrics’, *IEEE Transactions on Software Engineering* **27**(7), 630–650.
- Fenton, N. E. & Pfleeger, S. L. (1998), *Software Metrics: A Rigorous and Practical Approach*, PWS Publishing Co.
- Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1994), *Design Patterns*, Addison Wesley Publishing Company.
- Gosling, J., Joy, B., Steele, G. & Bracha, G. (2005), *The Java(tm) Language Specification*, third edn, Addison-Wesley.
- Harrison, R., Counsell, S. & Nithi, R. (2000), ‘Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems’, *Journal of Systems and Software* **52**, 173–179.
- Holub, A. (2003), ‘Why extends is evil: Improve your code by replacing concrete base classes with interfaces’, JavaWorld.com.
- Lorenz, M. & Kidd, J. (1994), *Object-oriented software metrics: a practical guide*, Prentice-Hall.
- Mikhajlov, L. & Sekerinski, E. (1998), A study of the fragile base class problem, in ‘ECOOP ’98: Proceedings of the 12th European Conference on Object-Oriented Programming’, Springer-Verlag, London, UK, pp. 355–382.
- Qualitas Research Group (2008), ‘Qualitas corpus. release 20080603’, <http://www.cs.auckland.ac.nz/~ewan/corpus/>.
- Steyaert, P., Lucas, C., Mens, K. & D’Hondt, T. (1996), Reuse contracts: managing the evolution of reusable assets, in ‘OOPSLA ’96: Proceedings of the 11th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications’, ACM, pp. 268–285.
- Tempero, E., Noble, J. & Melton, H. (2008), How do Java programs use inheritance? an empirical study of inheritance in Java software, in J. Vitek, ed., ‘22nd European Conference on Object-Oriented Programming’, Paphos, Cyprus, pp. 667–691.
- Venners, B. (2005), ‘Design principles from design patterns: A conversation with Erich Gamma, Part III’, <http://www.artima.com/lejava/articles/designprinciples4.html>.

# Modular Transactional Memory

Matthew Brecknell  
matthew@brecknell.net

Paul Roe  
p.roe@qut.edu.au

Queensland University of Technology, Australia

## Abstract

Software transactional memory has the potential to greatly simplify development of concurrent software, by supporting safe composition of concurrent shared-state abstractions. However, STM semantics are defined in terms of low-level reads and writes on individual memory locations, so implementations are unable to take advantage of the properties of user-defined abstractions. Consequently, the performance of transactions over some structures can be disappointing.

We present Modular Transactional Memory, our framework which allows programmers to extend STM with concurrency control algorithms tailored to the data structures they use in concurrent programs. We describe our implementation in Concurrent Haskell, and two example structures: a finite map which allows concurrent transactions to operate on disjoint sets of keys, and a non-deterministic channel which supports concurrent sources and sinks.

Our approach is based on previous work by others on boosted and open-nested transactions, with one significant development: transactions are given types which denote the concurrency control algorithms they employ. Typed transactions offer a higher level of assurance for programmers reusing transactional code, and allow more flexible abstract concurrency control.

**Keywords:** Software transactional memory, concurrency control, transactional boosting, open-nested transactions.

## 1 Introduction

Software transactional memory (Shavit & Touitou 1997) has emerged as a promising alternative to lock-based techniques for controlling concurrent access to shared data structures. STM provides an abstraction of physical memory, in which arbitrary sequences of memory operations can be executed as transactions, with the expected semantics of atomicity, consistency and isolation. STM saves the programmer from many of the pitfalls of lock-based programming, such as race conditions (failing to take the correct locks) and deadlocks (taking locks in the wrong order). Importantly, STM transactions are *composable* (Harris et al. 2005), whereas structures built using locks and condition variables typically are not. The latter has major benefits for the modularity and reusability of concurrent software.

Copyright © 2010, Australian Computer Society, Inc. This paper appeared at the Thirty-Third Australasian Computer Science Conference (ACSC2010), Brisbane, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 102. B. Mans and M. Reynolds, Eds. Reproduction for academic, not-for-profit purposes permitted provided this text is included.

Yet, if we care about performance, composability can be illusory. The semantics of transactional memory are typically defined by whether read and write operations on individual memory variables commute. While this is *sufficient* to ensure correctness under composition, it is not always *necessary*. Abstractions already limit the admissible sequences of memory operations, often in ways that would allow more efficient concurrency control. STM implementations are unable to take advantage of this fact, so the performance of concurrent abstractions can be worse than one might intuitively expect.

Transactional boosting (Herlihy & Koskinen 2008) and open-nesting (Ni et al. 2007) are methodologies which address this issue, allowing programmers to improve concurrency without losing composability. Boosting allows existing data structures to be used in transactional contexts, and open-nesting allows such data structures to be constructed using STM. Both allow programmers to replace low-level concurrency control on individual memory locations with higher-level *abstract* concurrency control.

Previous implementations providing boosted and open-nested transactions allow execution of arbitrary code in transactional contexts. This is necessary for abstract concurrency control, but it weakens the STM abstraction. The use of untyped transactions also limits the expressivity of abstract concurrency control, resulting in suboptimal implementations. In this paper, we develop a more disciplined approach, making the following contributions:

- We propose *types for modular transactional concurrency control*. Transactions are typed according to the concurrency control algorithms they employ, for improved assurance and flexibility of abstract concurrency control.
- We describe our implementation in Concurrent Haskell, combining transactional boosting and open-nested transactions with modular blocking and choice.
- Since our system uses a two-phase commit protocol internally, we provide a simple two-phase commit mechanism to users at no extra cost.

This paper is organised as follows. After some critical background (§2), we give an overview of our system (§3), some example applications (§4), and a discussion of future directions (§5). Finally, we draw conclusions (§6).

## 2 Background and related work

To make the work accessible to a general audience, we include the following background material. We introduce software transactional memory (§2.1), and show

how transactional boosting and open-nested transactions can be used to improve concurrency without losing composability. We also provide a brief overview of Concurrent Haskell (§2.2), since our system makes crucial use of certain features of the language.

## 2.1 Software transactional memory

STM is an abstraction of shared memory, in which arbitrary sequences of memory operations are executed as transactions, such that concurrent transactions *appear* to execute serially. Programmers can use familiar sequential reasoning within transactions, and only need to ensure that transactions take consistent states to consistent states.

For example, in a transaction to debit an account, we can be sure that concurrent updates will not modify the balance between the times we read and write the balance. In pseudo-code:

```
atomic debit(amount) {
    balance := balance - amount;
}
```

Although this appears like a critical section synchronised on the account, note that critical sections often do not compose correctly. For example, to atomically debit multiple accounts, it would be necessary to lock *all* accounts before *any* can be modified.

Transactions do compose naturally, and concurrent processes cannot observe intermediate states. For example, during the following transfer, no other process can observe a state where the amount is absent from both accounts:

```
atomic transfer(amount, debtor, creditor) {
    debtor.debit(amount);
    creditor.debit(-amount);
}
```

An **atomic** method may contain primitive operations (reads and writes) on transactional memory, as well as calls to pure functions and other **atomic** methods. An **atomic** method called from an ordinary method is executed as a top-level transaction, while an **atomic** method called from another **atomic** method is executed in the context of the calling transaction.

There is no need to specify locks; the runtime system determines which locks are required by observing the memory locations accessed during execution. No deadlock is possible, even when there are concurrent transfers in opposite directions; the system aborts and restarts transactions as necessary to ensure isolation and global progress.

### 2.1.1 Informal semantics

STM ensures *opacity*<sup>1</sup> of concurrent execution traces (Guerraoui & Kapalka 2008). Opacity is similar to the serialisability criteria used to describe database transactions, but also requires that even transactions which abort only observe consistent states. Informally, an execution trace is opaque, or serialisable, if it is *equivalent*, in a certain sense, to some sequential execution of the same transactions.

Equivalence of execution traces can be defined in terms of *conflicting* operations. Concurrent operations conflict when they cannot be reordered without potentially changing the outcome of some subsequent operation. For simple read-write memory variables, a write conflicts with any other operation on the same variable, while all other pairs of operations are without conflict.

<sup>1</sup>Most existing transactional memory implementations ensure opacity (Guerraoui et al. 2008), so opacity might be considered the *de-facto* standard semantics for STM.

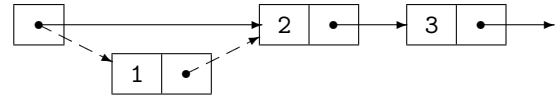


Figure 1: Deletion from the front of a linked list

Executions are conflict-equivalent<sup>2</sup> if they contain the same transactions, and the partial order among conflicting operations is the same. An implementation must ensure that if operations of one transaction conflict with those of another, then all the conflicting operations of one must come before the conflicting operations of the other.

### 2.1.2 The limits of composability

In a semantics defined by conflicts between read and write operations on simple memory variables, the notion of conflict is defined without reference to any semantics of variable *contents*. A write conflicts with any other operation on the same variable, irrespective of the value written. This provides a simple abstraction, but it can result in disappointing performance. Consider a simple integer map:

```
class map_stm {
    atomic int lookup(int key);
    atomic insert(int key, int value);
    atomic delete(int key);
}
```

In an abstract sense, operations on distinct keys can be reordered without affecting subsequent operations. However, an STM implementation of this structure is likely to generate conflicts between operations on distinct keys. This can easily be seen in a sorted linked list (figure 1), where deletion of the first node requires a write to the head, which necessarily conflicts with any other operation on the structure. Similar conflicts occur even in more sophisticated structures, such as hash tables and skip lists.

At the level of the operations themselves, such conflicts are unavoidable. The problem is that the impact of these conflicts on concurrency may leak far beyond the scope of those operations. For example:

```
atomic update(int key, map_stm src) {
    value := src.lookup(key);
    new_value := expensive_computation(key, value);
    src.insert(key, new_value);
}
```

Conflicts generated internally by `map_stm` operations mean that `update` operations on distinct keys might not be able to execute concurrently. That is, `update` operations are unnecessarily serialised, *including* any unrelated but expensive computations they contain.

To rectify this loss of composability, the notion of conflict must be generalised to include the operations of user-defined abstractions, for *abstract* concurrency control. The following sections describe two existing approaches.

### 2.1.3 Transactional boosting

Transactional boosting (Herlihy & Koskinen 2008) allows existing non-transactional data structures to be used in transactional contexts; for example, the containers in `java.util.concurrent`. Boosting is possible for structures where each operation has an inverse

<sup>2</sup>Opacity actually requires a slightly stronger notion of equivalence, but we find the notion of conflict useful for the discussion of abstract concurrency control which follows.



and is *linearisable*<sup>3</sup>, and where the commutativity of operations can be identified. Boosting makes use of *compensating actions* to undo the effects of operations when a transaction must abort, and *abstract locks* to ensure that non-commutative operations occur in a serialisable (opaque) order.

Our pseudo-code language is extended with new keywords, **boosted** and **onAbort**. A **boosted** method may be called from an **atomic** method, and contains non-transactional code to access the boosted structure and acquire abstract locks. Within a **boosted** method, **onAbort** may be used to record compensating actions to execute if the transaction is aborted. Abstract locks are acquired in a lock manager which holds locks until the acquiring transaction commits or aborts, and which aborts transactions as necessary to avoid deadlock.

Given an integer-indexed lock structure `lock_idx`, an existing linearisable integer map may be boosted as follows:

```
class map_boosted {
  map_linear map;
  lock_idx lock;
  //...
  boosted delete(int key) {
    lock.shared(key);
    value := map.lookup(key);
    if (value is not null) {
      lock.exclusive(key);
      onAbort { map.insert(key, value); }
      map.delete(key);
    }
  }
}
```

Operations on the underlying structure are already linearisable, and so do not need any further synchronisation. However, locks are required to ensure abstract serialisability among boosted transactions. Since operations on distinct keys commute, one lock per key is sufficient for this purpose. Further, shared locks can be used for read-only operations.

The benefit of boosting this structure is that operations like **update** (§2.1.2) can reach a higher degree of concurrency. Using existing highly concurrent data structures, the concurrency between individual map operations can be significantly improved. More importantly, abstract locking may also allow *other* parts of transactions to run with improved concurrency. If those other parts are expensive, the improvement can be dramatic.

In our framework (§3), typed transactions provide the same benefits as transactional boosting, but with greater flexibility. For example, typed transactions allow individual compensating actions to be aggregated into more efficient bulk operations.

### 2.1.4 Open-nested transactions

Open nested transactions (Ni et al. 2007) are nested transactions which commit as soon as they complete, making their effects visible to other processes before their parent transactions commit. Together with abstract locking, and compensating actions in case the parent transaction aborts, open-nested transactions provide similar benefits to transactional boosting.

In full generality, open-nested transactions do not have clear semantics. For example, there is no reasonable answer to the question of how an implementation

should behave when an open-nested transaction and its parent *both* modify the same memory location. We therefore consider a restricted usage, in which open-nested transactions are only used to implement abstractions in which:

- *every* operation which accesses internal state of the abstraction is an open-nested transaction,
- those operations *only* access the internal state of the abstraction, and
- internal state is *encapsulated*, and is not directly accessible from outside the abstraction.

Agrawal et al. (2008) give an ownership type system which formalises and enforces a similar, but more refined restriction.

The use of open-nested transactions is then very similar to transactional boosting, except that the underlying structure is implemented using STM transactions. One difference is that open-nested transactions may contain open-nested transactions as descendants, whereas boosted transactions cannot be nested.

## 2.2 Concurrent Haskell

We have developed our system in Concurrent Haskell (Peyton Jones et al. 1996), as it provides a useful laboratory for experimenting with new ways of writing imperative programs, primarily because side-effects are associated with types. For readers unfamiliar with the language, we provide a brief overview here.

Haskell is a *purely functional* programming language, in which all values are immutable and all functions are free of side effects. Concurrent Haskell provides explicit threads and synchronisation as an extension provided by the Glasgow Haskell Compiler.

Useful programs must have side effects, so Haskell contains an imperative *sublanguage* which is embedded as an abstract parametric datatype, `IO`. Values of `IO` types describe *actions* which may be executed. If executed, an action may have side effects, and may return a result to its calling action. For example, following are the type signatures for two members of the standard library:

```
getChar :: IO Char
putChar :: Char -> IO ()
```

These are read as follows: `getChar` is an action which, if executed, may have side-effects (in this case, reading a character from the terminal) and returns a character result; and `putChar` is a function which takes a character as its only argument and returns an action which, when executed, may have side-effects (printing the character) and returns void.

Actions may be composed using various combinators, or alternatively, using *do*-notation:

```
echo :: IO ()
echo = do { c <- getChar; putChar c; echo }
```

Actions are first-class values, like those of any other type. Purity demands that the mere construction or manipulation of `IO` actions does *not* cause them to be executed. Indeed, an action is only executed if it is the single distinguished `IO` action called `main`, or one of its composite parts.

Since `IO` can describe arbitrary side-effects, it can also describe mutable memory cells, with type `IORef`. Manipulation of mutable cells only occurs within `IO` actions, so there is no contradiction of purity. In these signatures, the identifier `a` is a *type variable*, indicating that these functions, and `IORef`, have parametric polymorphic types:

<sup>3</sup>Linearisability is a property of individual instances of an abstract data type. Informally, an object is linearisable if each operation appears to take effect “instantaneously at some point between its invocation and response” (Herlihy & Wing 1990). In this context, linearisability is a requirement on the data structure underlying a boosted transaction, while opacity and serialisability are global properties of the transactional system.

```
newIORef :: a -> IO (IORef a)
readIORef :: IORef a -> IO a
writeIORef :: IORef a -> a -> IO ()
```

Concurrent threads are started by `forkIO`:

```
forkIO :: IO () -> IO ThreadId
```

Synchronisation is provided by `MVar`, which acts as a simple one-place blocking channel:

```
newEmptyMVar :: IO (MVar a)
putMVar :: MVar a -> a -> IO ()
takeMVar :: MVar a -> IO a
```

`IO` also provides an exception handling mechanism.

### 2.2.1 Haskell STM

Concurrent Haskell includes an STM implementation (Harris et al. 2005). It is unique in its provision of a static guarantee that transactions only contain operations on memory, without requiring special language support.

Like `IO` actions, STM transactions are described by an abstract datatype, `STM`. As for `IO`, constructing an STM transaction does not cause its execution. A transaction can only be executed as an `IO` action, via a call to `atomically`:

```
atomically :: STM a -> IO a
```

Primitive transactions operate on transactional memory variables of type `TVar`:

```
newTVar :: a -> STM (TVar a)
readTVar :: TVar a -> STM a
writeTVar :: TVar a -> a -> STM ()
```

The `do`-notation is overloaded so transactions may be composed. For example, the `debit` function (§2.1) is written:

```
debit :: Int -> TVar Int -> STM ()
debit amount account = do
  balance <- readTVar account
  writeTVar account (balance - amount)
```

Another primitive transaction provides a general and composable form of conditional blocking. A transaction which calls `retry` blocks until at least one variable previously read by the transaction has been modified by another thread, and then aborts and restarts. It can be used to wait for arbitrary conditions. For example, the following transaction waits for clear funds before making a debit:

```
debitWhenClear :: Int -> TVar Int -> STM ()
debitWhenClear amount account = do
  balance <- readTVar account
  if amount <= balance
    then debit amount account
    else retry
```

Deterministic choice is provided by `orElse`, which takes two transactions, and runs the first, unless it would block with a call to `retry`, in which case it runs the second. For example, the following will debit the first of two accounts to have clear funds:

```
debitFirst :: Int -> TVar Int -> TVar Int -> STM ()
debitFirst amt act1 act2 = debitWhenClear amt act1
  'orElse' debitWhenClear amt act2
```

Note one of the benefits of a pure type system: transactions and `IO` actions have distinct types, so the type rules ensure that transactions only contain operations which can be performed atomically. Our work aims to realise similar benefits for abstract concurrency control.

### 2.2.2 Type classes

Haskell type classes<sup>4</sup> provide a disciplined approach to overloading. For example, following is a type class for equality comparison:

```
class Eq a where
  (==) :: a -> a -> Bool
```

This defines a new type class `Eq` with an overloaded operator method (`==`). The type variable (`a`) stands for each type which will instantiate the class, and must appear somewhere in the type of each method. A type instantiates a class with an `instance` declaration which provides a suitable implementation for each method, specific to that type. For example, `Bool` equality can be defined by *pattern-matching* on the constructors of the type:

```
instance Eq Bool where
  True == True = True
  False == False = True
  _ == _ = False
```

The `Eq` type class allows functions which work for all types supporting equality. For example, to test if an item is contained in a cons-list, by pattern-matching on the constructors for the list type:

```
elem :: Eq a => a -> [a] -> Bool
elem x (y:ys) = (x == y) || elem x ys
elem x [] = False
```

Here, the first part of the type signature (`Eq a =>`) indicates that the type variable (`a`) is *constrained*. That is, `elem` is only defined for types supporting equality. Square brackets around a type (`[a]`) indicate a list of that type; a single colon (`x:xs`) constructs a cons-cell which prepends an item to an existing list; and empty square brackets (`[]`) construct an empty list.

Instances may be defined in terms of simpler instances. For example, equality for lists is defined in terms of equality of elements:

```
instance Eq a => Eq [a] where
  (x:xs) == (y:ys) = (x == y) && (xs == ys)
  [] == [] = True
  _ == _ = False
```

This enables a form of *type-directed* programming.

## 3 Modular transactional memory

This section introduces our system. The design is inspired by Haskell STM, with support for abstract concurrency control. Transactions are attributed with descriptive types, to reconcile the need for arbitrary side-effects during abstract concurrency control, with the benefits of static guarantees provided by a pure type system. Typed transactions can also improve the expressivity and efficiency of abstract concurrency control.

### 3.1 Typed transactions

We begin by describing transactions and how they can be composed, without reference to any particular transactional data structures. A new type parameter is added to the type of transactions, to represent the concurrency control algorithms used, giving the type (`Tx t a`) of transactions using algorithms described by type `t` and returning type `a`.

In this context, an *algorithm* is the concurrency control (locking, compensating actions, etc.) for some abstract structure. For example, if `Mem` (defined elsewhere) is the type representing the concurrency control for ordinary transactional variables, then we can

<sup>4</sup>Haskell type classes are almost entirely unlike the classes of object-oriented programming. In particular, they are *not* types.

form the type `(Tx Mem a)` of transactions over ordinary variables. Indeed, the latter corresponds to the type `(STM a)` in the existing Haskell STM.

`Tx` is defined as an `I0` action, which executes in the context of a central transaction log of type `Log` (§3.3), and a *local* transaction log of type `t`:

```
data Result a = Abort | Retry | Result a

newtype Tx t a =
  Tx { runTx :: Log -> t -> IO (Result a) }
```

The `Result` type is used internally to distinguish between transactions which abort and restart due to a conflict, those which call `retry`, and those which return a result normally.

Thus, if concurrency control algorithms are types, then their values are *local transaction logs*, specialised to process transactions over the corresponding data structures. For composition of transactions, a transaction log must provide certain operations, including initialisation, nesting and finalisation. These requirements are represented as a type-class:

```
class TLog t where
  run :: (t -> Undo -> Commit -> IO a) -> IO a
  nest :: t -> (Undo -> IO a) -> IO a
```

`TLog` methods are called by the framework, which passes an `IO` action as a callback. A method implementation should execute the callback, passing the required parameters back to the framework. Typically, method implementations will also wrap the callback with resource management and exception handlers.

The type `Commit` describes the actions to perform when a transaction commits, and will be explained later (§3.4). The type `Undo` describes an action which reverses the effects of a transaction when it aborts, and is a simple type alias:

```
type Undo = IO ()
```

The framework calls `run` to begin execution of a transaction. The implementation of `run` must initialise a new transaction log (type `t`), as well as global abort and commit actions (types `Undo` and `Commit`) to pass back to the body. The framework then calls `nest` to obtain a local undo action for every call to `catch` and `orElse`.

An instance of `TLog` is sufficient to implement all of the core combinators of Haskell STM, including overloading the `do`-notation, and these functions:

```
catch :: (TLog t, Exception e)
=> Tx t a -> (e -> Tx t a) -> Tx t a

retry :: TLog t => Tx t a
orElse :: TLog t => Tx t a -> Tx t a -> Tx t a

atomically :: TLog t => Tx t a -> IO a
```

For example, the following shows the implementation of `catch`, which presents nested transactions as an exception-handling mechanism. Here, backslash (`\`) introduces a lambda (anonymous function):

```
catch body handler = Tx (\l t -> nest t (\undo ->
  Control.Exception.catch (runTx body l t)
    (\e -> do { undo; runTx (handler e) l t } )))
```

Haskell programmers might be alarmed at the pervasive use of `IO` in the definitions of `TLog` and `Tx`. Do we fail to take advantage of the benefits of a pure type system? We do not believe so, because every use of `IO` in the definition of a transactional data structure is labelled with the type `t` of the local transaction log. Thus, transaction types act as *certificates of origin*, which can help a user to understand the nature of some transactional code, without the need to study all of its parts.

### 3.2 Combining transaction types

The real value of this framework is in its support for composing transactions over arbitrary *combinations* of data structures, with abstract concurrency control. Since we can only compose transactions of the same local log type, we provide a type-level operator `(:&)` to construct *combined* local log types, and a mechanism to inject transactions into these combined types. For example, if `Map` is the type representing concurrency control for associative maps, then we can form the type `(Tx (Mem :& Map) a)` of transactions over ordinary variables *and* associative maps. The operator is implemented as a simple right-associative pair type, which may be right-nested to arbitrary depth:

```
data l :& r = l :& r
infixr 5 :&
```

It is straightforward to provide a `TLog` instance for a combined type, based on `TLog` instances for the component types:

```
instance (TLog l, TLog r) => TLog (l :& r)
```

It is also easy to implement the following explicit injections, which could be used to compose two transactions over different structures, in sequence:

```
injl :: Tx l a -> Tx (l :& r) a
injrr :: Tx r a -> Tx (l :& r) a
```

However, explicit injection functions are cumbersome to use, so we define an inclusion relation which automates injections. This is implemented as a multi-parameter type class `(<)` with suitable instances for combined types, following Swierstra (2008):

```
class t :< c where
  inject :: Tx t a -> Tx c a
```

For example, given transactions over `Mem` and `Map`, with the following types:

```
example_mem_basic :: Tx Mem ()
example_map_basic :: Tx Map ()
```

We can `inject` into an unspecified, but constrained, combined transaction type, resulting in transactions with the following types:

```
example_mem :: (Mem :< c) => Tx c ()
example_mem = inject example_mem_basic

example_map :: (Map :< c) => Tx c ()
example_map = inject example_map_basic
```

These can then be composed in sequence, resulting in a transaction with the following type:

```
example :: (Mem :< c, Map :< c) => Tx c ()
example = do { example_mem; example_map }
```

The type can be read: `example` is a transaction over any set of structure types containing *at least* `Mem` and `Map`. This `example` transaction could be further composed with other transactions, with new constraints added to the transaction type as necessary.

Finally, at the point of execution, the transaction must be given a concrete type, to complete the automatic injection:

```
example_exec =
  atomically (example :: Tx (Mem :& Map) ())
```

By combining abstractions this way, we assume that the areas of memory accessed by those abstractions are non-overlapping. This is a reasonable assumption, provided abstractions are properly encapsulated. It is also consistent with the restriction on the use of open-nested transactions (§2.1.4).

### 3.3 Primitive transactions

So far, we have described transactions and how they compose in a very general sense, without saying anything about how they are implemented. In particular, we have not shown how to provide primitive transactions for any structure of interest.

At this point, we are forced to make some significant implementation choices. There are numerous approaches to STM implementation, with equally many trade-offs (Larus & Rajwar 2007). Many approaches suit certain workloads, but not others. Since we are still in the early stages of developing this framework, our choices have been determined primarily by the need for simplicity, and secondly by a bias toward long transactions and high-contention workloads, since we expect this to be the area where abstract concurrency control will provide the greatest benefit. We intend to revisit these choices as we continue development.

We have opted to implement transactions with *visible readers*. Whenever a transaction reads state from a data structure, it must register with that structure to receive notifications when updates by other transactions invalidate that observation. For serialisability, we abort any transaction which receives such a notification before it is ready to commit. Processor cache performance can be inhibited by visible readers, but we currently accept this for the sake of simplicity.

The system allows both *deferred* and *direct* updates, although support for direct updates is limited. In a deferred update, a transaction records its intention to update to its local transaction log, but the actual update is deferred until the transaction commits. Deferred updates only require locks to be acquired during the commit process, but read operations must consult the transaction log before the data structure in memory. During a direct update, a transaction immediately acquires a lock on the structure, records the previous state in its transaction log (in case the transaction aborts), and writes the update directly to the structure. Transactions using direct updates must take care to avoid deadlock, usually by aborting the transaction if a lock cannot be acquired within a short time limit. Currently, there is no support for deadlock detection, lock pre-emption or cascading aborts for direct updates.

A primitive transaction for a structure is created using the `primitive` function, which has the following type:

```
primitive :: Cert t -> (Log -> t -> IO (Result a))
          -> Tx t a
```

`Cert` is a type family used to restrict the construction of a primitive transaction to the module which defines the corresponding structure. Constructors for values of type `(Cert t)` should be private to the module which defines type `t`. This parameter thus certifies that the calling module is authorised to define a transaction of the given type.

The second argument provides the implementation for the primitive transaction. It is an `IO` action with access to the central transaction log (type `Log`) and the local transaction log (type `t`) for the respective structure type, and which returns a `Result` type. This means that a primitive transaction may return an ordinary result, or alternatively, `Abort` or `Retry`. Returning `Abort` causes the transaction to abort, for example, if a lock for a direct-mode update could not be acquired. `Retry` indicates that the primitive transaction should behave like a call to `retry`.

The local transaction log (of type `t`) is defined entirely by the module providing that transaction type. Typically, it records updates, in either deferred or direct mode. Examples given later (§4) show how local

transaction logs can improve the expressiveness of abstract concurrency control.

The central transaction log (type `Log`) is used to implement the visible reader protocol. A transaction which observes the state of a structure should register its `Log` with that structure. Concurrent transactions which update the structure must notify any registered transactions which are invalidated by those updates, by calling `invalidate`, which has type:

```
invalidate :: Log -> IO ()
```

To maximise the benefit of abstract concurrency control, the implementation of a transactional data structure should aim to send notifications only to transactions whose observations have in fact been invalidated, according to the abstract semantics of the structure.

For transactions over multiple structure types, it is important that invalidation is handled centrally, since an update to any previously observed structure could invalidate the transaction. The implementation of `primitive` checks that the transaction has not been invalidated after executing the body of each primitive transaction.

Conveniently, this mechanism also serves to implement conditional blocking. A transaction which calls `retry` simply waits for invalidation, and then aborts and restarts.

### 3.4 Commit actions

As a transaction executes, each local log accumulates updates in deferred or direct mode. When finished, it must either commit or abort. It can only commit if it can acquire suitable locks for deferred updates, and if it has not been invalidated by an update performed by another transaction.

The abort process is straightforward. Deferred-mode updates can simply be discarded. Direct-mode updates must be reverted, but this can be done safely, since locks have already been acquired. Locks must then be released, and reader registrations should be cleared.

The commit process is more complicated, and proceeds in the following phases:

1. Locks must be acquired for all deferred updates across all local transaction logs. If it is not possible to acquire all locks, then the transaction must abort without any updates being committed.
2. The transaction must be validated, with a check the the transaction has not been invalidated by updates committed by another transaction. Otherwise, it must abort.
3. Only if the transaction has not aborted, then deferred updates must be written across all local transaction logs, and notifications sent to invalidated transactions.
4. In any case, locks must be released, and reader registrations cleared.

We refer to the first and third phases as the *prepare* and *update* phases, respectively. They are performed by local transaction logs, while the second (validation) phase is performed centrally.

The system collects actions to execute for the prepare and update phases from local transaction logs, via the `run` method of the `TLog` class (§3.1). The actions are described by the `Commit` data type:

```
data Commit = Commit {
  prepare :: IO () -> IO (),
  update  :: IO ()
}
```

To begin the commit process, the framework constructs a chain of calls to `prepare` and `update` for all local logs involved in the transaction, such that each call to `prepare` is passed a callback consisting of any remaining calls to `prepare`, followed by validation and a sequence of `update` actions. Each `prepare` action should execute its callback *only* if it successfully acquires all locks necessary to ensure that the corresponding `update` can execute correctly. When the callback returns, the transaction has either aborted or committed, so `prepare` may release locks it previously acquired.

### 3.5 User-level two-phase commit

With the internal use of two-phase commit, it is possible to provide a form of two-phase commit directly to the user, without any additional cost. We provide `twophase`, with the following type:

```
twophase :: TLog t => Tx t (IO a) -> IO a
```

This is similar to `atomically`, but instead of taking a transaction returning an ordinary value, `twophase` takes a transaction returning an `IO` action. We refer to this as the *validate* action.

When executing a transaction, `twophase` performs the validate action between the prepare and update phases of the commit protocol. This guarantees that the transaction can commit, but there is still the option to abort before updates become visible to other transactions.

We define the operation of `twophase` as follows: if the validate action returns normally, then the transaction commits, and `twophase` returns the same result as the validate action; if the validate action throws an exception, then the transaction aborts, and `twophase` propagates the exception.

For example, the following requires confirmation from the user before debiting an account:

```
debitConfirm amount account = twophase (do
  balance <- readTVar account
  writeTVar account (balance - amount)
  return (do
    confirm <- prompt ("Balance = " ++ show balance)
    if not confirm
      then throw RejectDebit
      else return ()))
```

Without `twophase`, it would not be possible to ensure that the balance shown to the user is the same balance to which the debit is applied.

### 3.6 Blocking open-nested transactions

Open-nested transactions require special treatment, if they are to be allowed to include calls to `retry`.

Specifically, if an open-nested transaction blocks with a call to `retry`, and the parent transaction is invalidated by an update committed by another transaction, then the open-nested transaction should return control to the parent, to allow it to restart. The converse is not necessary: if an open-nested transaction is directly invalidated, but its parent has not been invalidated, then only the open-nested transaction need restart.

Further, if an open-nested transaction calls `retry` when the parent transaction is in the first branch of the choice operator (`orElse`), then the open-nested transaction should not block, but should immediately abort and return control to the parent, to allow it to execute the alternate `orElse` branch.

These requirements are satisfied by `open`, which has the following type:

```
open :: TLog t => Log -> Tx t a -> IO (Result a)
```

Like `atomically`, `open` executes a transaction, but requires an additional parameter for the parent transaction `Log`. The implementation of `open` uses this to register with the parent, to receive its invalidation signals, and to determine whether the parent is in the first branch of `orElse`. An `open`-nested transaction will only block if the parent is not in the first branch of `orElse`.

A call to `open` returns a `Result` type. `Retry` is returned when the parent is in the first branch of `orElse`, and the `open`-nested transaction calls `retry`. `Abort` is returned when the parent transaction has been invalidated. This result should be returned to the parent transaction (via `primitive`), after performing appropriate abstract concurrency control.

## 4 Example structures

This section describes several example transactional structures. First, we develop the prototypical structure, the simple transactional variable, and then contrast this with a boosted associative map structure. Finally, we develop a non-deterministic channel using open-nested transactions. Readers should bear in mind that the combining operators provided by the framework (§3.2) allow transactions over these structures to be composed together.

### 4.1 Simple transactional variables

In this framework, simple transactional variables are defined as any other structure. We define the structure itself, its local transaction log, and its primitive transactions. Using deferred-mode updates, the `TVar` structure may be defined as follows:

```
data TVar a = TVar {
  tvar_data :: IORef a,
  tvar_lock :: MVar (),
  tvar_readers :: CList Log,
}
```

Recalling that `IORef` is a mutable variable, and `MVar` provides synchronisation, the purpose of the first two fields should be clear. The third registers transactions which have observed the value of the `TVar`, for notification of updates. `CList` is a list type which supports concurrent insertion and deletion, and is necessary because many transactions may attempt concurrent reads from a `TVar`. The local transaction log can then be defined as follows:

```
data Write = forall a. Write (TVar a) a
```

```
data Mem = Mem {
  mem_write :: IORef [Write],
  mem_read :: IORef [CNode Log]
}
```

The `mem_write` field records write operations in deferred mode, and is used during subsequent read operations, as well as the commit process. Each record includes the `TVar` written to, and the value written. The `mem_read` field records reader registrations for all variables read during the transaction, and is used to clear those registrations at the end of the transaction. A `CNode` is simply a handle to a `CList` node.

The local transaction log, `Mem`, must be made an instance of `TLog`, so we must provide implementations for `run` and `nest`. Of these, `nest` is the most interesting:

```
nest mem body = do
  saved_writes <- readIORef (mem_write mem)
  body (writeIORef (mem_write mem) saved_writes)
```

For a nested transaction, the framework calls `nest` with the local transaction log (`mem`) and the `body` of the nested transaction. First, `nest` saves a copy of the write log, and then calls the `body`, passing an undo action which may be used to restore the saved write log in the event that the nested transaction aborts. Recall that constructing an IO action does not cause it to be executed immediately. If needed, the framework will execute the undo action at the appropriate time; otherwise it will be discarded.

Implementations of primitive transactions must certify their authority to create a transaction of the required type. A private type definition, declared as a `Cert` instance, serves this purpose:

```
data MemCert = MC
type instance Cert Mem = MemCert
```

The `readTVar` primitive may now be defined:

```
readTVar :: TVar a -> Tx Mem a
readTVar var = primitive MC (\log t -> do
  look_aside <- write_log_lookup var t
  case look_aside of
    Just value -> return (Result value)
    Nothing -> readDirect log t var)
```

In case the transaction has previously performed an uncommitted write to the same variable, a read operation first examines the write log. If the `TVar` is not found, `readDirect` is called to insert the transaction `Log` into the `tvar_readers` field of the `TVar`, add the registration to the `mem_read` field of the local transaction log, and return the current value of the variable. To ensure reads are properly sequenced with respect to writes in other transactions, `readDirect` takes a lock on the `TVar`.

The `writeTVar` primitive simply adds a `Write` entry to the `mem_write` field of the local transaction log, or replaces an existing entry for the same `TVar`. Then, at commit time, the `prepare` action attempts to acquire locks for all `TVars` in the write log:

```
prepare callback = do
  write_log <- readIORef (mem_write t)
  foldr tryWithLock callback write_log
```

Here, `foldr` reduces the write log with `tryWithLock`, which takes a `Write` entry and a callback, and executes the callback only if it acquires a lock on the corresponding `TVar`. If `prepare` succeeds, `update` writes each new value in the write log to the corresponding `TVar`, and invalidates any transactions which have registered reads on each `TVar`.

The abort action is empty, but the `run` method clears reader registrations which have accumulated in the `mem_read` field of the local transaction log, at the end of the transaction.

## 4.2 Finite map

The map structure presented here is the moral equivalent of the boosted map shown earlier (§2.1.3), since it consists of an underlying linearisable data structure wrapped with abstract concurrency control. However, there are substantial differences in our method: the typed local transaction log is accessible to read operations, and therefore supports deferred-mode updates; it also allows us to aggregate operations on each map structure, for a more efficient commit process.

In fact, this implementation is very similar to that of simple transactional variables, so we only show key differences. Unfortunately, there are no readily available concurrent map implementations for Haskell, so we use a purely functional map (shown as `M.Map` below) protected by a single lock. Although not concurrent, it is linearisable, and so still meets the requirements for boosting. Despite the absence of operation-level concurrency, abstract concurrency control is still

valuable, since it may increase concurrency between other parts of transactions.

The `TMap` structure is defined as follows. The important difference is that `tmap_readers` is defined on a *per-key* basis, so that operations on distinct keys do not conflict.

```
data TMap k v = TMap {
  tmap_data :: IORef (M.Map k v),
  tmap_lock :: MVar (),
  tmap_readers :: IORef (M.Map k (CList Log))
}
```

This complicates the process of clearing reader registrations, which must ensure that `tmap_readers` does not accumulate empty registration lists. The local transaction log therefore stores each reader registration with its associated key and `TMap`:

```
data Write = forall k v.
  Write (TMap k v) (M.Map k (Maybe v))

data Read = forall k v.
  Read k (TMap k v) (CNode Log)

data Map = Map {
  mem_write :: IORef [Write],
  mem_read :: IORef [Read]
}
```

Note that each `Write` entry aggregates all the updates for a particular map, with deletions represented by a null value.

Primitive transactions, nesting and finalisation are similar to those for simple transactional variables, so we do not reproduce them here.

## 4.3 Non-deterministic unordered channel

This section describes an application of open-nested transactions, including those which block (§3.6), as well as direct-mode updates.

More importantly, it also demonstrates the value of typed transactions to the *consumer* of transactions. To improve concurrency, we allow this structure to exhibit a higher level of non-determinism than is generally allowed by the semantics of closed word-based transactional memory. With this structure, different executions of the same transactions may yield different results, even if the serialisations are the same.

We believe that controlled non-determinism has a useful role in concurrent programming, because it can provide higher efficiency and concurrency. Nevertheless, it is a fundamental change in semantics, so transactions which may exhibit this behaviour should be clearly identified. Our framework achieves this by ensuring that every transaction which operates on this structure has a type which includes its local log type.

The `NDChan` structure, with local transaction log type `NDC`, is defined as an unordered, unbounded collection with operations `put` and `take`, where `put` inserts an item, and `take` either blocks or returns an item which it removes from the collection:

```
put :: NDChan a -> a -> Tx NDC ()
take :: NDChan a -> Tx NDC a
```

The operation of `take` is non-deterministic, so it may choose any element from the collection, and may even block (`retry`) when the collection is not empty. We do want the structure to be useful, so we insist that `take` only blocks when there are *concurrent* transactions performing `take`, or the collection is empty.

Note that non-deterministic behaviour cannot be localised, but may infect the whole transaction. For example, since `take` may `retry` instead of returning an item, any `orElse` containing a `take` becomes a non-deterministic choice.

The underlying structure, `TChan`, is built with simple transactional variables. This is just the multicast channel described by Harris et al. (2005), with the important parts reproduced here:

```
type Link a = TVar (Node a)
data Node a = Empty | Full a (Link a)

data TChan a = TChan {
  take_end :: TVar (Link a),
  put_end :: TVar (Link a)
}

putTChan :: TChan a -> a -> Tx Mem ()
putTChan chan item = do
  link <- readTVar (put_end chan)
  empty <- newTVar Empty
  writeTVar link (Full item empty)
  writeTVar (put_end chan) empty

takeTChan :: TChan a -> Tx Mem a
takeTChan chan = do
  link <- readTVar (take_end chan)
  node <- readTVar link
  case node of
    Empty -> retry
    Full item next -> do
      writeTVar (take_end chan) next
      return item
```

While `putTChan` and `takeTChan` only conflict with each other if the channel is empty, each will always conflict with itself. This limits the concurrency between transactions performing the same operation on this structure, even if most of the work in those transactions is performed elsewhere. This problem is similar to the `update` operation described earlier (§2.1.2), and the solution is the same. While we cannot increase the concurrency between individual `TChan` operations, abstract concurrency control may allow *other* parts of transactions to run concurrently.

The non-deterministic specification of `take` affords a lot of freedom. We implement `put` in deferred-mode, and `take` as a direct-mode open-nested transaction. As abstract operations, `put` and `take` do not conflict with themselves, nor each other, with one exception: a `put` conflicts with a `take` which has blocked. This is the only situation in which we need to register a read operation. We do need to record items extracted by `take`, so that they may be returned if the transaction aborts. `NDChan` therefore contains a `TChan`, and a list of reader registrations:

```
data NDChan a = NDChan {
  ndchan :: TChan a,
  ndchan_read :: CList Log
}
```

The local transaction log (type `NDC`) records items extracted by `take`, deferred `put` operations, and reader registrations:

```
data Buffer = forall a. Buffer (NDChan a) [a]

data NDC = NDC {
  ndc_take :: IORef [Buffer],
  ndc_put :: IORef [Buffer],
  ndc_read :: IORef [CNode Log]
}
```

Here, a `Buffer` is an association between a channel and a list of items of the same type.

The `put` primitive inserts a buffer to the `ndc_put` field of the local transaction log, or augments an existing buffer if the `NDChan` is already present.

The `take` primitive first searches the `ndc_put` field of the local transaction log. If `take` finds a buffer for the required `NDChan`, it removes and returns an item from that buffer. Otherwise, `take` executes an open-nested transaction to extract an item from the underlying `TChan`, and buffers the item in the `ndc_take` field of the local transaction log:

```
take :: NDChan a -> Tx NDC a
take chan = primitive NDCert (\log t -> do
  lookAside <- extract_put chan (ndc_put t)
  case lookAside of
    Just item -> return (Result item)
    Nothing -> do
      result <- open log (takeTChan (ndchan chan))
      case result of
        Abort -> return Abort
        Retry -> do
          node <- cinsert log (ndchan_read chan)
          modifyIORef (ndc_read t) (node:)
          return Retry
        Result item -> do
          buffer_take chan (ndc_take t) item
          return (Result item))
```

Careful readers will observe a race condition between `open` and the reader registration in the `Retry` branch, which may allow a `put` in another transaction to go unnoticed. Our current solution is to rerun `open` after registering the read, but we omit this for brevity. In a future revision, it may be possible to improve our implementation of open-nested transactions, such that this is not necessary.

The remaining work is done in the abort and commit actions. To abort a transaction, items buffered in `ndc_take` must be returned to their associated channels, and readers registered on those channels must be invalidated:

```
unbuffer :: Buffer -> IO ()
unbuffer (Buffer chan items) = do
  atomically (mapM_ putTChan (ndchan chan) items)
  invalidate_clist (ndchan_read chan)

abort = do
  take_buffers <- readIORef (ndc_take t)
  mapM_ unbuffer take_buffers
```

Note, we use `atomically` rather than `open`, since the abort action must not fail.

For commit, the prepare action is empty; it simply calls the body it is passed. The update action is similar to the abort action, inserting items buffered in `ndc_put` to their associated channels, and invalidating registered readers.

Finally, `nest` performs similar local manipulations of `ndc_put` and `ndc_take` buffers.

This example demonstrates that we can not only raise the level of abstraction of concurrency control, but we can relax and redefine its semantics to achieve improved concurrency. A typed local transaction log provides the necessary flexibility, acting as a medium for primitive operations to communicate with each other.

## 5 Discussion

Our vision, like that of other work on abstract concurrency control for transactional memory, is for concurrent software which reflects the structure of the application domain. Programming with locks and condition variables risks catastrophic structural failure as a system grows and requirements change. In contrast, composable memory transactions provide a safe foundation for evolution of concurrent software, but concurrency may suffer as transactions become more complex. Adding abstract concurrency control allows hot-spots to be removed with *local* changes only, and without changing the overall semantics of the program. Typed transactions provide a particularly flexible approach to abstract concurrency control.

Our system is still in the early stages of development, and much remains to be done. So far, our focus has been raising the level of abstraction for transactional programming, without sacrificing concurrency. This focus will continue as we experiment

with new data structures and concurrent programming patterns, but must eventually shift to other concerns.

We are yet to examine performance. Our current implementation is high-level and naïve, so we do not yet expect good absolute performance, nor to compete with other current STM implementations. Nevertheless, we believe that typed local transaction logs should provide the means to generalise many of the techniques used in high-performance STM implementations, to support abstract concurrency control. Some of the techniques we are interested in exploring include dependence-aware transactional memory (Ramadan et al. 2009) and *aggressive* transactional boosting (Koskinen & Herlihy 2008). We would also like to improve support for lock-based approaches (Dragojević et al. 2009), and contention management (Guerraoui et al. 2005). It remains to be seen whether it is possible (let alone desirable) to support multiple approaches to concurrency control in the one framework.

For many structures, like the finite map example (§4.2), development is likely to be similar to that of simple transactional variables. A natural evolution of the framework would be to factor these common parts into a library implemented in low-level C. Co-operation with the run-time thread scheduler may be beneficial for some parts, such as lock management.

Our development and presentation has so far been informal. Given the pitfalls which plague concurrent software, it will be particularly important to formalise our approach. This is not just to verify the soundness of the framework, but to provide correctness criteria to guide the development of new transactional structures. The formalisations given by Herlihy & Koskinen (2008) and Agrawal et al. (2008) give us confidence that this should be possible for a broad class of data structures. Our semantics would also need to account for modular blocking (`retry`) and choice (`orElse`) in the presence of open-nested transactions, and should accommodate structures with relaxed semantics, such as the non-deterministic channel (§4.3).

Certain features of the Haskell language are crucial to our design: its purity, and its ability to embed imperative sublanguages as abstract datatypes. Could our approach work in other languages? We think so, but probably only with language extensions. Thus, while our approach may only be accessible to Haskell programmers for the time being, we hope that in the longer term, it can contribute to the development of future concurrent programming languages. In particular, we believe that our work adds weight to arguments in favour of type systems which capture side-effects, particularly mutation, in a compositional way.

## 6 Conclusion

We have demonstrated our approach to abstract concurrency control, which allows concurrent data structures to be composed safely, without the loss of concurrency which affects transactional memories based on simple transactional variables. Our primary contribution is that transactions should be typed according to the structures they manipulate. Typed transactions admit abstract concurrency control without completely abandoning static safety guarantees. Typed local transaction logs also provide more flexible, and potentially more efficient abstract concurrency control.

## References

Agrawal, K., Lee, I.-T. A. & Sukha, J. (2008), Safer open-nested transactions through ownership, *in*

‘Principles and Practice of Parallel Programming (PPoPP ’08)’, ACM, New York, pp. 291–292.

Dragojević, A., Guerraoui, R. & Kapalka, M. (2009), Stretching transactional memory, *in* ‘Programming Language Design and Implementation (PLDI ’09)’, ACM, New York, pp. 155–165.

Guerraoui, R., Henzinger, T. A., Jobstmann, B. & Singh, V. (2008), Model checking transactional memories, *in* ‘Programming Language Design and Implementation (PLDI ’08)’, ACM, New York, pp. 372–382.

Guerraoui, R., Herlihy, M. & Pochon, B. (2005), Toward a theory of transactional contention managers, *in* ‘Principles of Distributed Computing (PODC ’05)’, ACM, New York, pp. 258–264.

Guerraoui, R. & Kapalka, M. (2008), On the correctness of transactional memory, *in* ‘Principles and Practice of Parallel Programming (PPoPP ’08)’, ACM, New York, pp. 175–184.

Harris, T., Marlow, S., Peyton Jones, S. & Herlihy, M. (2005), Composable memory transactions, *in* ‘Principles and Practice of Parallel Programming (PPoPP ’05)’, ACM, New York, pp. 48–60.

Herlihy, M. & Koskinen, E. (2008), Transactional boosting: a methodology for highly-concurrent transactional objects, *in* ‘Principles and Practice of Parallel Programming (PPoPP ’08)’, ACM, New York, pp. 207–216.

Herlihy, M. P. & Wing, J. M. (1990), ‘Linearizability: a correctness condition for concurrent objects’, *ACM Transactions on Programming Languages and Systems* **12**(3), 463–492.

Koskinen, E. & Herlihy, M. (2008), ‘Aggressive transactional boosting’. Submitted.

Larus, J. R. & Rajwar, R. (2007), *Transactional Memory*, Synthesis Lectures on Computer Architecture, Morgan and Claypool.

Ni, Y., Menon, V. S., Adl-Tabatabai, A.-R., Hosking, A. L., Hudson, R. L., Moss, J. E. B., Saha, B. & Shpeisman, T. (2007), Open nesting in software transactional memory, *in* ‘Principles and Practice of Parallel Programming (PPoPP ’07)’, ACM, New York, pp. 68–78.

Peyton Jones, S., Gordon, A. & Finne, S. (1996), Concurrent haskell, *in* ‘Principles of Programming Languages (POPL ’96)’, ACM, New York, pp. 295–308.

Ramadan, H. E., Roy, I., Herlihy, M. & Witchel, E. (2009), Committing conflicting transactions in an stm, *in* ‘Principles and Practice of Parallel Programming (PPoPP ’09)’, ACM, New York, pp. 163–172.

Shavit, N. & Touitou, D. (1997), ‘Software transactional memory’, *Distributed Computing* **10**(2), 99–116.

Swierstra, W. (2008), ‘Data types à la carte’, *Journal of Functional Programming* **18**, 423–436.



# Memory Efficient State-Space Analysis in Software Model-Checking

Anshuman Mukherjee

Zahir Tari

Peter Bertok

School of CS & IT, RMIT University  
GPO Box 2476V, Melbourne, VIC 3001, Australia  
Email: {amukherj,zahirt,pbertok}@cs.rmit.edu.au

## Abstract

*Formal methods* have an unprecedented ability to endorse the correctness of a system. In spite of that, it has been limited to safety-critical and mission-critical systems owing to significant time and memory costs involved. Lately, our ever increasing dependency on software in all walks of our life has necessitated using formal methods for a wider range of softwares. In this paper, we propose an algorithm to make this possible by reducing the memory requirement for *model checking*, a widely used formal method. A model-checker stores all explored states in memory to ensure termination. The proposed algorithm slash memory costs by storing these states in compressed form. In compressed form, a state is stored as how different it is from its previous state. Our experiments report a memory reduction of 95% with only doubling of computation delay. Aforesaid reduction allows model checking in a machine with only a fraction of memory needed otherwise. Consequently the advantage is twofold, 1)enormous savings as only a small physical memory is required and 2)as more states can now be stored in a memory of same size, the chances of complete state-space analysis is exceedingly high.

**Keywords:** State-space compression, Model checking, Formal methods, State-space explosion

## 1 Introduction

Traditionally, a software is considered “fail-safe” if it has passed a rigorous testing phase(Beizer 1990). However, the crash of Ariane 5 launcher(Clarke et al. 2000) and the deaths due to malfunctioning of Therac-25 radiation therapy machine(Rushby 1989) in spite of rigorous software testing suggest otherwise. The team investigating these accidents recommended(Clarke et al. 2000, Rushby 1989) using formal methods(FM) to complement testing as the former assures exhaustive verification of a system. However, FM have a high price-tag attached due to *state space explosion*(Christensen et al. 2001) problem, which compel developers to completely skip FM in order to meet software budget. Considering our dependence on software in everyday life(e.g. traffic signals, elevators), skipping FM amounts to risking millions of human lives. In this paper, we propose methods of reducing the cost of FM so that they are more widely used.

Copyright ©2010, Australian Computer Society, Inc. This paper appeared at the Thirty-Third Australasian Computer Science Conference (ACSC2010), Brisbane, Australia. Conferences in Research and Practice in Information Technology (CR-PIT), Vol. 102. B. Mans and M. Reynolds, Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.

Model Checking(MC)(Clarke et al. 2000) is a formal method which verifies a concurrent system automatically. A model checker requires formal design of the system and the properties to be verified. It then explores the state-space of the system to find a state<sup>1</sup> which violates the given properties, where state-space is the set of states reachable from initial state. If a violating state is found, it is returned as a counterexample. Otherwise, the model checker returns ‘yes’, implying that the properties are satisfied by all reachable states of the system.

During state-space exploration, there might be states generated more than once. To prevent analysing the same states repeatedly for desired properties, it is necessary to remember the states already explored by storing them in memory. This also ensures termination, a condition where no new states are generated. However, model checking is plagued with state-space explosion problem, often resulting in gigantic number of states. This causes manyfold increase in memory costs, as each new state has to be stored. Such bottlenecks in available memory hinders model checking.

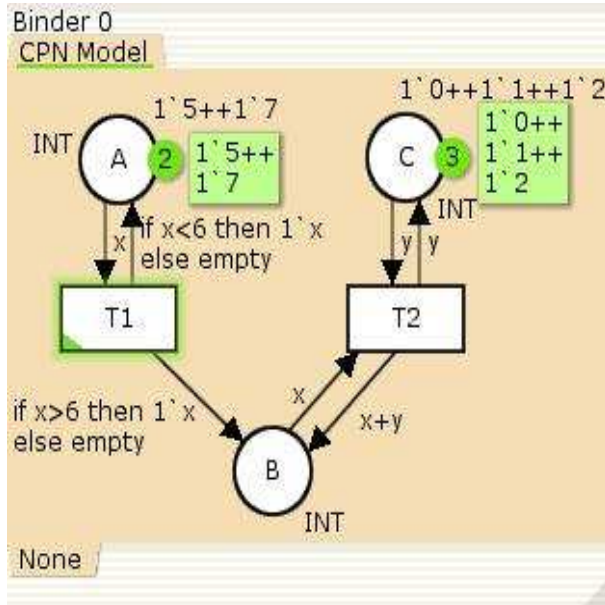
Some solutions based on ‘Partial storage’ address the problem by storing only a subset of explored states. Although this reduces memory requirement, it is difficult to decide the set of states to be deleted. If a deleted state is reached again in future, it is treated as a new state and explored further. The proposed solution has no such issues as it uses ‘Exhaustive storage’ in which all explored states are compressed and stored in a suitable data structure(e.g. hash-table). The states need to be decompressed before comparison as it is possible for more than one state to have similar compressed state.

In this paper, we devise a novel method to reduce the memory costs otherwise involved in model checking. We propose storing states in difference form, instead of explicit form, resulting in reduced memory requirements. In difference form, a state is stored as how different it is from its previous states. We propose an algorithm, known as **Sequential algorithm**, to explore the state-space by storing states in difference form.

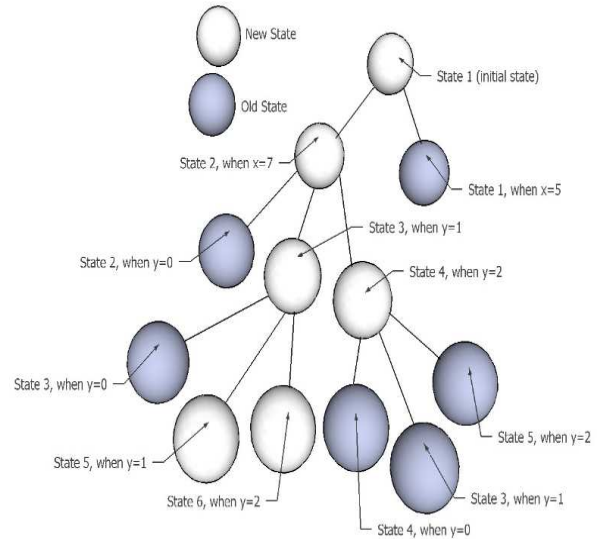
Our contributions can be summarised as:

1. We propose an algorithm to reduce the memory requirement for model checking by storing states in compressed form. The results indicate upto 95% reduction in memory requirement.
2. It is possible for many different explicit states to have the same difference state. Therefore, we propose an algorithm to decompress the states before comparison. Our decompression algorithm only doubles the time needed to generate the state-space. This is 33% lower than the time taken by (Evangelista & Pradat-Peyre 2005).

<sup>1</sup>‘Marking’ is sometimes used synonymously to a state



(a) A Coloured Petri-Net model. Variables  $x$  and  $y$  are of type INT



(b) A part of reachability graph for CPN model in Figure 1(a)

Figure 1: A Coloured Petri-net model and its reachability graph.

The remainder of the paper is organised as follows. Section 2 introduces state-space analysis and Coloured Petri-nets. In section 3 the sequential algorithm is proposed. We tabulate and plot the experimental results in section 4 and discuss the outcome in section 5. We look at the related work in section 6 and conclude in section 7.

## 2 Background

Model checking involves three basic steps: 1) Modeling the system, 2) Specifying the properties to be verified and 3) Verifying the properties in all reachable states of the model. The first step requires creating a formal representation of the system. The representation depends on model checking tool to be used for verification in step 3. Some common languages for system representation are PROMELA for SPIN (*Basic Spin Manual* 2007), C programming language for BLAST (*Blast Manual* 2008) and Coloured Petri-Nets (CPN) for CPN Tools (*CPN Tools* 2009). Due to subtle differences between these representation languages, it is difficult to propose a generic algorithm for memory reduction. In this paper, the proposed algorithm specifically target CPN models. However, we do not claim any advantage in using CPN models. The proposed algorithm is based on the assertion that “Change in a state is smaller than the state itself” and our algorithm will work for all representation languages, as long as this assertion holds. The assertion is valid because systems usually change in many small steps rather than a single large step. Experiments report a 95% reduction in memory, which further endorse our assertion.

We now define CPN and briefly explain state-space analysis using an example.

### 2.1 Coloured Petri-Nets

Coloured Petri-Nets (Jensen 1997) are Petri-Nets extended with programming constructs. The concurrent constructs of Petri-Nets are supplemented with data-definition and data manipulation constructs of programming languages. CPN is used for design, specification, simulation and verification of systems.

In contrast to PN, each token in CPN has an attached data value. The datatype of this value determine the colour of token. All tokens in a place must be of the colour specified by the colour set of that place.

Figure 1(a) shows a CPN model with 3 places (the circles) and 2 transitions (the rectangles). Place A has 2 tokens (indicated by 2 in the circle next to it)  $1'5$  and  $1'7$ , where  $1'5$  implies that there is 1 token with integer value 5. The tokens in a place are listed next to it and they are separated by ‘++’ symbol. The text “ $1'0++1'1++1'2$ ” near place C implies that it has 1 token with value 0, 1 token with value 1 and 1 with value 2. Each place also has its colour (or datatype) inscribed next to it. In this model, all places have colour INT and hence they can only have integer tokens.

A place and a transition are connected by an arc. Transition T1 in Figure 1(a) has an input arc from place A and two output arcs to places A and B. When T1 executes, it removes tokens from input place A and adds tokens to output places A and B. The tokens removed/added is found by evaluating the arc inscription. As all these arc inscriptions are defined in terms of variable  $x$ , they can be evaluated by assigning an appropriate value to  $x$  and this is called *binding of  $x$* . The binding of  $x$  is decided after considering the tokens in input place and the inscription of input arc. In case of T1, input place A has tokens  $1'5$  and  $1'7$  while the input arc inscription is  $x$ . Hence the only possible values of  $x$  for which T1 is enabled are 5 and 7. When T1 fires with  $x$  bound to 5, token  $1'5$  is removed from A and added back to A. No token is added to place B as the if condition is not satisfied. When T1 fires with  $x$  bound to 7, token  $1'7$  is removed from A and added to place B. The bindings for which T2 is enabled can be determined identically.

### 2.2 State-Space Analysis

In this section, we discuss the problem in detail and outline the proposed solution. State-space analysis of a model is done by generating a reachability graph. Each model has a unique initial state and this is represented by the root node of a reachability graph. At its initial state, the system might have a set of enabled

events which can bring in a change in state. Each of these events are represented by a separate edge from the root node of the reachability graph and lead to a new node representing the new state. These new states are then analysed for the set of enabled events. For each enabled event, an outgoing edge is added to the corresponding node. This in turn generates another set of new states to be analysed identically. This analysis continues till the set of new states have no enabled event.

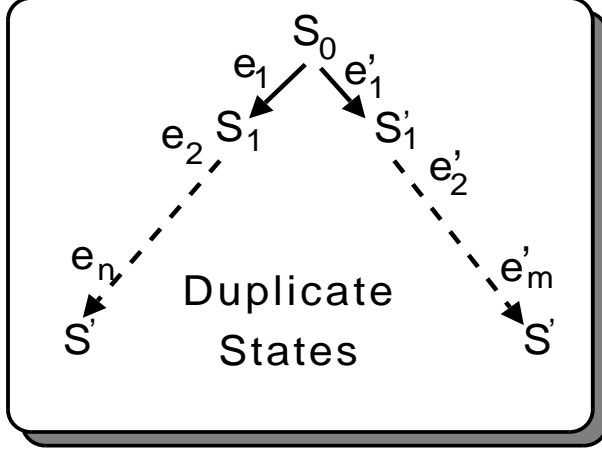


Figure 2:  $S'$  reached using two different sequential of events from  $S_0$

However, it might be possible to reach a state from the initial state by executing different sequence of events. Suppose  $S_0$  is the initial state of a model  $M$  and let  $S'$  be a state reached by the following two sequence of events

$$S_0[e_1]S_1[e_2]S_2[e_3] \cdots [e_m]S' \\ S_0[e'_1]S'_1[e'_2]S'_2[e'_3] \cdots [e'_n]S'$$

where  $\forall i \in [1, m]: e_i$  and  $\forall j \in [1, n]: e'_j$  are events and  $S_{i-1}[e_i]S_i$  denotes that event  $e_i$  in state  $S_{i-1}$  leads to state  $S_i$ . This is shown in Figure 2. If  $\exists i \in [1, n]: (i < m) \wedge (e_i \neq e'_i)$ , the state  $S'$  can be reached using two different sequence of events and therefore it has a *duplicate state*. The reachability graph for  $M$  will have two nodes representing the same state  $S'$ . However, analysing both the nodes and their children (each of which will also have a duplicate node) is a waste of resources. Larger the number of duplicate nodes for a state, greater the wastage in resources. Furthermore, if there exists a non-empty sequence of events  $[e_1 e_2 \cdots e_r]: r > 0$  that cause no net change in state of a model  $M$ , the model checker might never terminate. This is shown in Figure 3. Let  $S$  be some state of model  $M$  and  $\forall i \in [1, r]: e_i$  be events such that

$$S[e_1]S_1[e_2]S_2[e_3] \cdots [e_r]S \\ \text{or } S[e_1 e_2 e_3 \cdots e_r]S$$

Such state of affairs would lead to analysis of the set of states  $\{S, S_1, S_2, \cdots, S_{r-1}\}$  forever and state-space analysis might never finish. Consequently, it is nec-

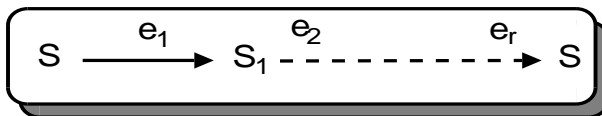


Figure 3: The sequence of events  $[e_1 e_2 \cdots e_r]$  causes no net change in state

essary to remember the states already explored and

ignore any duplicate states encountered. A model-checker remembers explored states by storing them in memory. When a state is generated during state-space exploration, it is compared with the stored states to determine if it is new or duplicate of a previously generated state. If it is a duplicate state, the corresponding node in reachability graph becomes a terminal node and it is not analysed any further. Otherwise, the new state is stored in memory and is analysed for enabled events. However, due to state-space explosion, large amount of memory is needed to store all unique states. In this paper, we propose an algorithm to reduce the memory requirement by storing a state as how different it is from its previous state.

At any time, there might be thousands of explored states stored in memory. Comparing each state generated with all stored states might take long. Hence the states are stored in a hash-table to ensure constant time lookup.

We illustrate the problem using an example. Figure 1 shows a Coloured Petri-Net model and a part of its reachability graph. All duplicate nodes in Figure 1(b) are shaded. Initially, the CPN model has 2 tokens in place A and 3 in C. This is represented by State 1 in Figure 1(b) and being the root node of reachability graph, it is stored in memory. The enabled events at this state are  $(T1, x=5)$  and  $(T1, x=7)$ , where T1 is the enabled transition and  $x=5$  or 7 is the binding for which it is enabled. Corresponding to these two enabled events, the root node in Figure 1(b) has two outgoing edges, one for each event. When T1 fires with  $x=5$ , there is no change in state as all tokens remain in their previous places and the edge corresponding to this event leads to a shaded node in Figure 1(b). As this node represents a duplicate state, it is not analysed any further. The other event  $(T1, x=7)$  results in moving a token from A to B, leading to State 2. Being a new state, it is represented using a bright node in Figure 1(b). Furthermore, it is stored and further analysed for enabled transitions. State 2 has three enabled events:  $(T2, y=0)$ ,  $(T2, y=1)$  and  $(T2, y=2)$ . The first event causes no change in state and is represented by a shaded node in Figure 1(b). The other two events change the value of token in B leading to State 3 and State 4 and these are represented by bright nodes in Figure 1(b). Being new states, they are stored in memory and further analysed for enabled events. Remaining states are explored analogously to generate the complete reachability graph.

The reachability graph in this example has infinite number of states. Other models might have finite number of states. However, the number of states is almost always gigantic leading to state-space explosion (Clarke & Berezin 1998). Complete state-space analysis is possible only when the available memory ( $\alpha_A$ ) in a machine is at least equal to memory needed to store all unique states in reachability graph ( $\alpha_M$ ) of model  $M$ . Otherwise, if  $\alpha_A < \alpha_M$ , only a partial state-space analysis can be performed and the analysis stops when memory is full. We propose an algorithm for compact representation and storage of a state. Using sequential algorithm, the memory needed to store all unique states in reachability graph of a model  $M$  reduces to  $\alpha'_M$ . This allows 1) Complete reachability analysis in a machine with a smaller memory  $\alpha'_A$  if  $\alpha'_A \geq \alpha'_M$ , where  $\alpha'_A < \alpha_A$  and  $\alpha'_M < \alpha_M$  2) Even when  $\alpha_A < \alpha'_M$ , the partial state-space analysis can have at least a few more steps. With available memory remaining same, we are able to create a reachability graph with more states due to less memory needed to store a state.

### 3 Proposed Algorithm for Memory Efficient State-Space Analysis

In this section, sequential algorithm is proposed for memory efficient state-space analysis. The algorithm proposed in this section specifically target CPN models. However, we do not claim of any advantage in using CPN models. As stated previously, the proposed algorithm is based on assertion that “Change in a state is smaller than the state itself” and as long as this assertion holds, the proposed algorithm is valid for all modeling languages. For CPN, a change in state occurs if one or more tokens either 1) move to another place, 2) change their value, 3) are created in the model, 4) are deleted from the model or a combination of these such that the colour of each token match the colour-set of containing place. In a CPN model, these changes are brought in by a transition. However, a transition usually modifies the place and value information of only a small number of tokens. Furthermore, a very small number of tokens are usually created or deleted by a transition. For example transition T1 in Figure 1(a) fires with  $x=7$ , it only changes the place of token 1'7. Therefore it is substantially cheaper to store a state S as how different it is from its previous state. Based on this, we propose sequential algorithm to generate memory efficient reachability graph in next section.

#### 3.1 Sequential Algorithm

In this section, we introduce sequential algorithm to reduce the memory requirements of model checking. Such a reduction will increase the affordability and consequent use of model checking in software development. The focus of sequential algorithm is storing states in difference form which is defined as:

**Definition 1** The difference form of a state  $S_{st}$ , with previous state  $S_{pv}$ , is defined as the changes necessary in  $S_{pv}$  to generate  $S_{st}$  and it is denoted as  $D_{st}$ . If  $D_{st}$  is the difference form of a state and  $S_{pv}$  is its previous state, the state  $S_{st}$  can be regenerated in explicit form as  $S_{st} = S_{pv} + D_{st}$ .

An explicit state has information for all tokens and is often referred as state in this paper, omitting the adjective ‘explicit’. We illustrate how to find the difference form of a state using an example.

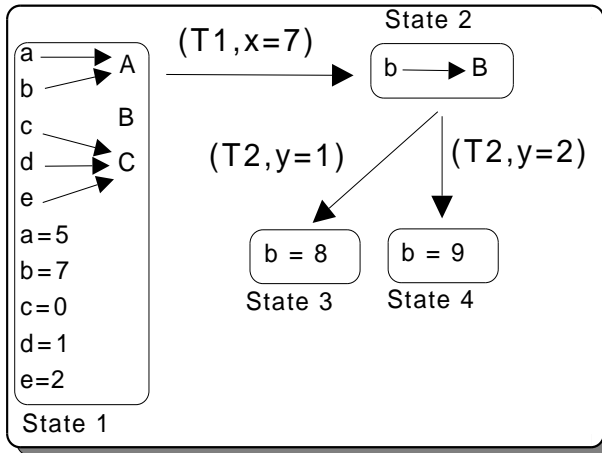


Figure 4: A part of reachability graph in Figure 1(b) using sequential algorithm

Figure 4 presents a portion of reachability graph for the CPN model in Figure 1(a) using sequential algorithm. Initially the model is in State 1 and since it does not have a previous state, it is stored in explicit

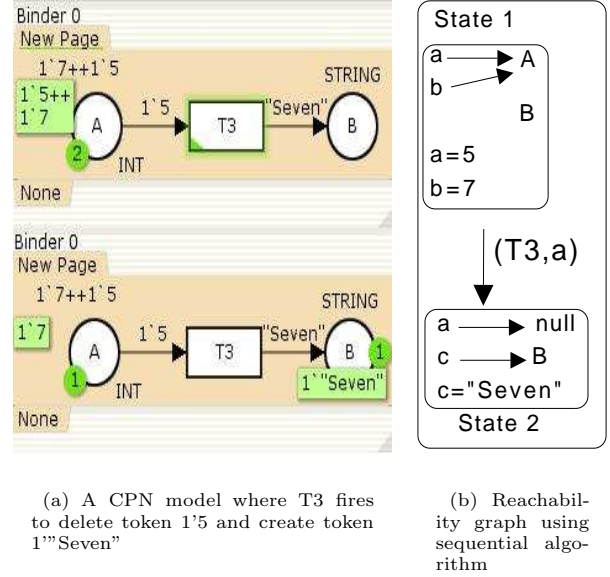


Figure 5: State change when tokens are created and/or deleted

form in Figure 4. Each token in the model is given a name in Figure 4. Furthermore, its place is assigned by an arrow ( $\rightarrow$ ) and value is assigned by an equal ( $=$ ) symbol. For example, the token  $1'5$  in place A of Figure 1(a) is named **a**. Its place is assigned as  $a \rightarrow A$  while the value is assigned as  $a=5$ . Similarly, token  $1'7$  in A is named **b** and assigned place and value as  $b \rightarrow A$  and  $b=7$ . All other tokens are named arbitrarily and assigned place and value accordingly.

When the event  $(T1, x=5)$  occurs, the model persists in State 1. As a result, the difference form is empty (or null) and not drawn in Figure 4. However,  $(T1, x=7)$  takes it to State 2. In order to store the new state in difference form, we need to find the changes in State 1 brought by this event. We find that the event moved token  $1'7$  to place B. This information is sufficient to construct State 2 from State 1. We therefore store State 2 in difference form as  $b \rightarrow B$ .

The event  $(T2, y=1)$  in State 2 lead to State 3. Likewise, the event  $(T2, y=2)$  lead to State 4. In order to store these new states in difference form, the changes in State 2 brought by these events need to be found. On inspecting these events, both are found to change the value of token in place B. While  $(T2, y=1)$  changes the value to 8,  $(T2, y=9)$  changes it to 9. Given State 2 in explicit form, this information is sufficient to construct State 3 and State 4. Accordingly, State 3 is stored as  $b=8$  while State 4 is stored as  $b=9$ . The difference form for other states are calculated identically. Additionally, each state also store a pointer to its previous state. This is necessary to regenerate the states as explained later. As evident from this example, it takes less space to store states in difference form.

A state change also occurs when an event creates or deletes one or more tokens. If an event deletes a token **a**, the new state can be represented in difference form by assigning the place for **a** as  $null(a \rightarrow null)$ . Similarly when an event creates a new token, it is given an arbitrary name and assigned the corresponding place and value information. This is illustrated by an example in Figure 5. The CPN model in Figure 5(a) has a transition T3 which removes token  $1'5$  from place A and adds  $1'Seven$  to place B. Considering the value in latter token, place B is assigned colour-set STRING. The reachability graph of the model using sequential algorithm is presented in Figure 5(b). The tokens in place A are assigned names **a**

and b. Initial state of the model is stored explicitly in Figure 5(b) as there is no previous state to calculate difference. When event (T3,1'5) occurs, it deletes the token a and creates a new token which we name c. The new state can be stored in difference form by assigning the place of a to null and assigning the place and value information for newly created token. Given State 1 in explicit form, aforesaid information is sufficient to regenerate State 2. This example further endorse a reduction in memory requirement when states are stored in difference form.

In this section, we explained how to obtain the difference form of a state and demonstrated the memory reduction when states are stored in difference form. However, more than one explicit state can produce the same difference state. This necessitates converting states into explicit form before comparison. This is explained with an example in next section.

### 3.1.1 Expanding a State in Difference Form

In this section, we demonstrate backtracking in order to revert a difference state. This is necessary for comparison as more than one explicit state can produce the same difference state.

When a state is generated during state-space exploration, it has to be compared with stored states to determine if it is new. However, compressing and comparing it with states stored in difference form might lead to an error owing to multiple states having the same difference form. Supposing three states  $S_a$ ,  $S_b$  and  $S_c$  produce the same difference form  $D_{abc}$ . When either of the three states is encountered for the first time,  $D_{abc}$  is stored in memory. When the other two states are encountered and compared with stored states in compressed form, they are wrongly interpreted as duplicate state. Therefore it is essential to revert a stored state before comparing. Such a conversion is called *expanding* and is done by *backtracking*.

**Definition 2** *Backtracking is the process of regenerating a state by recursively adding the most recent changes for each token to its previous state until a state stored in explicit form is reached. If  $S_n$  and  $D_n$  are the explicit and difference states at depth  $n$  of a reachability graph, the former can be obtained from latter using a backtracking function BK, where  $S_n = BK(D_n) = D_n + S_{n-1}$ . Since  $S_0$  is always in explicit form, this equation can be solved for all  $n \leq h$ , where  $h$  is the height of the reachability graph.*

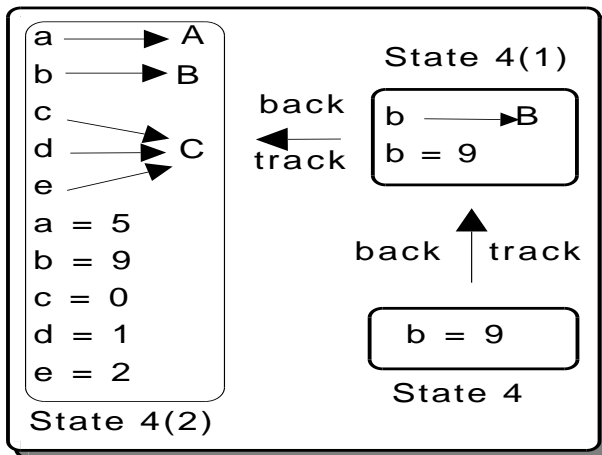


Figure 6: Backtracking to expand State 4 in Figure 4

We illustrate backtracking with an example. State 4 is stored in difference form in Figure 4 and in order

to expand it, we need to backtrack till an explicit state is encountered, as shown in Figure 6. Initially, State 4 contains new value for token b and from Definition 1, we should get State 4 in explicit form by updating its previous state(which we hope is in explicit form) with this value. However, on backtracking one step in Figure 4, we reach State 2 which is also stored in difference form. But it gives additional information about the place of token b. We add the place information from State 2 with value information from State 4 and get a meta-state 4(1) shown in Figure 6. We call 4(1) a meta-state as it was obtained by combining two different states. Using Definition 2, State 4 can now be obtained by updating the previous state of State 2 with information in metastate 4(1). On further backtracking, State 1 is encountered which is actually in expanded form. We update it with the information in 4(1) and get another metastate 4(2). By Definition 2, 4(2) is State 4 in expanded form.

As backtracking is an additional overhead when states are stored in difference form, they increase the time needed for state-space analysis. Definition 2 requires backtracking to initial state  $S_0$  for expanding each state. However, this leads to large delay with increase in height of reachability graph. In the next section, we discuss ways for reducing this problem.

### 3.1.2 Decreasing the Cost of Expanding

In this section, we discuss ways of reducing the additional delay incurred while backtracking. Reducing this delay will reduce the overall time for model checking.

So far we have stored only the initial state in explicit form while all other states to be stored in difference form. Although this ensures maximum reduction in memory requirement, Definition 2 will require backtracking to the initial state for expanding. As a result, the states far from initial state take long to expand.

In order to reduce the delay, the number of backtracking steps need to be minimised. If every state at depth  $i \delta: i \in \mathbb{N}^2$  steps from initial state is stored in expanded form, expanding a state will never need a backtracking greater than  $\delta-1$  steps. Therefore starting from initial state,  $\delta^{th}$ ,  $2\delta^{th}$ ,  $3\delta^{th} \dots$  states are stored in expanded form. The algorithm can be tuned by accepting different values of  $\delta$ . Finally, we propose our algorithm next section.

### 3.1.3 Proposed Algorithm

In this section, the proposed algorithm is introduced and explained. As specified previously, sequential algorithm achieves memory reduction by storing states in difference form. Starting from the initial state, it explores remaining reachable states of the model using depth-first search(DFS) algorithm(Cormen et al. 2001). Each explored state is stored in a hash-table.

When a state is generated, a hash-function is used to find its index in hash table. If this index is empty, the state is new. Its difference form is calculated and inserted at this index. Furthermore, the enabled transitions are identified and one of them is fired. Otherwise, if there are states already stored at this index, they are all expanded and compared with the generated state. If there is no match, the state is new and it is inserted at the head of list at this index. Additionally, one of its enabled transitions is executed. In case of a match, the state is duplicate of a state analysed previously. It is neither stored nor analysed for enabled transitions.

<sup>2</sup> $\mathbb{N}$  is the set of natural numbers starting from 0



The proposed algorithm calculates the difference form of a state by comparing it with its previous state. However to reduce delay in backtracking, all states at depth  $i \delta: i \in \mathbb{N}$  from initial state are stored in explicit form, where  $\delta$  is the shortest distance between two explicit states. In order to expand a state, the algorithm implements backtracking until an explicit state is encountered.

The proposed algorithm also implements two-level hashing at an index if the number of states stored at that index exceeds a threshold. In our algorithm, we set threshold as  $M/10$ , where  $M$  is an estimation of the total number of reachable states. When two level hashing is used, the index of primary hash table contains hash-function for secondary hash table.

The proposed sequential algorithm has three parts:

1. **SEARCH:** The steps are listed in Algorithm 1. This algorithm accepts a state( $S_{st}$ ), it's previous state( $S_{pv}$ ) and the distance of  $S_{st}$  from last expanded state(depth) as input. A hash function  $H$  is used to find the index for state  $S_{st}$  as shown in step 1. The algorithm then checks the content of hash-table at this index. There can be three possibilities.

---

**Algorithm 1:** SEARCH (State  $S_{st}$ , int depth, State  $S_{pv}$ )

---

**Data:** current state  $S_{st}$ , steps away from last explicit state(depth), previous state  $S_{pv}$   
**Result:** Decide if a state generated is new

```

1  $i \leftarrow H[S_{st}]$ ;
2 if  $HASH[i] = NULL$  then
3   INSERT( $S_{st}$ , depth,  $S_{pv}$ );
4   foreach  $S'$  such that  $S_{st}[(t, c)] S'$  do
5     SEARCH( $S'$ , (depth+1) mod  $\delta$ ,  $S_{st}$ );
6 else if  $HASH[i]$  points to a linked list then
7   foreach state  $D$  in linked list do
8     if  $D$  is in difference form then
9        $D \leftarrow RECONSTRUCT(D)$ ;
10    if  $D = S_{st}$  then return;
11  end
12  INSERT( $S_{st}$ , depth,  $S_{pv}$ );
13  foreach  $S'$  such that  $S_{st}[(t, c)] S'$  do
14    SEARCH( $S'$ , (depth+1) mod  $\delta$ ,  $S_{st}$ );
15 else if  $HASH[i]$  contains a hash function then
16    $H' \leftarrow HASH[i]$ ;
17    $j \leftarrow H'[S_{st}]$ ;
18   if  $HASHi[j]$  is empty then
19     INSERT( $S_{st}$ , depth,  $S_{pv}$ );
20     foreach  $S'$  such that  $S_{st}[(t, c)] S'$  do
21       SEARCH( $S'$ , (depth+1) mod  $\delta$ ,  $S_{st}$ );
22   else
23     if  $HASHi[j]$  is in difference form then
24       |  $HASHi[j] \leftarrow RECONSTRUCT(HASHi[j])$ ;
25     end
26     if  $HASHi[j] = S_{st}$  then return;
27     INSERT( $S_{st}$ , depth,  $S_{pv}$ );
28     foreach  $S'$  such that  $S_{st}[(t, c)] S'$  do
29       SEARCH( $S'$ , (depth+1) mod  $\delta$ ,  $S_{st}$ );
30   end
31 end

```

---

- (a) *Hash table contain NULL at this index:* In this case, it is the first time this state is generated. Hence Algorithm 2 is called to store the state at this index. Any enabled event at this state is fired. Steps 2-4 in Algorithm 1 check and handle this case.

- (b) *Hash table contain a linked-list at this index:* In this case, each state in the linked-list has hashed to this index.  $S_{st}$  is compared with each state in this list. If a state is stored in difference form, it is expanded before comparison using Algorithm 3. In case of a match, the state is neither stored nor analysed for an enabled event. Otherwise, the state is stored at the head of linked list using Algorithm 2 and an enabled event is fired. Steps 5-11 in Algorithm 1 check and handle this case.

- (c) *Hash table contain a hash-function at this index:* If this case, all states which hashed to this index are stored in a separate hash-table  $HASHi$  indexed by the function  $H'$  stored at this index. In step 14, the index in second hash table is calculated using this hash function. Step 15 checks if this index is empty or has a state stored. In case this index is empty or does not contain this state, it is inserted at this index using Algorithm 2 and its enabled events are fired. Otherwise the algorithm returns. Steps 15-24 in Algorithm 1 handle these cases.

---

**Algorithm 2:** INSERT (State  $S_{st}$ , int depth, State  $S_{pv}$ )

---

**Data:** current state  $S_{st}$ , steps away from last explicit state(depth), previous state  $S_{pv}$   
**Result:** Insert state  $S_{st}$  into hash table

```

1 if depth=0 then
2   new.type  $\leftarrow$  explicit;
3   new.state  $\leftarrow S_{st}$ ;
4 else
5   new.type  $\leftarrow$  difference;
6   new.state  $\leftarrow S_{st} - S_{pv}$ ;
7   new.prev  $\leftarrow S_{pv}$ ;
8 end
9  $i \leftarrow H[S_{st}]$ ;
10 if  $HASH[i] = NULL$  then
11    $HASH[i] \leftarrow$  new;
12 else if  $HASH[i]$  points to a linked list then
13   insert new at the head of linked list;
14   if length(linked list)  $\geq |M|/10$  then
15     foreach state  $d$  in linked list do
16       if  $d$  is in difference form then
17         |  $d \leftarrow RECONSTRUCT(d)$ ;
18       end
19       add  $d$  to  $HASHi[H'[d]]$ ;
20     end
21    $HASH[i] \leftarrow H'$ 
22 end
23 else if  $HASH[i]$  points to a hash function then
24    $H' \leftarrow HASH[i]$ ;
25    $j \leftarrow H'[S_{st}]$ ;
26    $HASHi[j] \leftarrow$  new;
27 end

```

---

2. **INSERT:** This algorithm is responsible for inserting a state into hash table and is listed in Algorithm 2. It accepts a state( $S_{st}$ ), it's previous state( $S_{pv}$ ) and the distance of  $S_{st}$  from last expanded state(depth) as input. The fields of a pointer "new" are assigned the required values before storing it in appropriate index. Based on the value of delta, the state is either stored in explicit or difference form and this is assigned to 'type' field of pointer 'new'. In case of former, the explicit state  $S_{st}$  is assigned to 'state' field of

‘new’. Otherwise the difference, given by “ $S_{st} - S_{pv}$ ” is assigned to ‘state’ field. Additionally, in latter case, a pointer to previous state is stored in ‘prev’ field of ‘new’. This is shown in steps 1-8 of Algorithm 2.

The index of hash-table at which this state is to be stored is calculated in step 9. There could be three possible cases:

- Hash table contain NULL at this index:* This is the case when  $S_{pv}$  is generated for the first time. The contents of pointer ‘new’ is simply copied to this index of hash-table. This is shown in steps 10-11 of Algorithm 2.
- Hash table contain a linked-list at this index:* In this case, the contents of ‘new’ is copied to head of linked list. Furthermore, it is checked if the list contains more than 10% of an estimated total number of states. In that case, the states in this linked list is stored in another hash table and the hash function is stored at this index. This is done in steps 12-22 of Algorithm 2.
- Hash table contain a hash-function at this index:* In this case, the hash-function stored at this index is used to find the index in secondary hash-table and the contents of pointer ‘new’ is copied to that index. Steps 23-26 in Algorithm 2 handle this case.

- RECONSTRUCT:** This algorithm accepts a difference state and expands it by backtracking. The steps are listed in Algorithm 3. In steps 1-4, the algorithm backtracks and add each state encountered until an explicit state is reached. Finally, the state  $D_{st}$  in explicit form can be calculated by adding the sum to the explicit state encountered. This is shown in step 5.

---

**Algorithm 3:** RECONSTRUCT(State  $D_{st}$ )

---

**Data:** State  $D_{st}$  in difference form

**Result:** Expanded form of D is returned

```

1 while d.type=difference do
2   sum=sum+d.state ;
3   d=d.prev;
4 end
5 return d+sum;
```

---

### 3.1.4 Complexity Analysis

The proposed sequential algorithm promises to reduce the amount of space necessary to store the states by using difference states. However, this reduction is accompanied by a delay due to backtracking. In this section, we calculate the reduction provided and derive the time needed for extra processing.

Let  $\delta$  be the distance between two expanded states. We pointed out earlier that the initial state is stored in expanded form. Other states in expanded form are those at depth  $\delta$ ,  $2\delta$ , and so on. If a reachability graph has height  $n$ , the depth of last expanded node is  $\lfloor \frac{n}{\delta} \rfloor * \delta$ .

Assuming that the average number of new states generated by a transition is  $k(>1)$ , the number of states at depth  $d$  is given by  $k^d$  and this is illustrated in Figure 7. All dark circled represent explicit states while shaded circles represent difference states. The number of expanded states in a reachability of height  $n$  is the sum of the number of expanded states at depth  $0, \delta, 2\delta, \dots, \lfloor \frac{n}{\delta} \rfloor * \delta$ . This is given by

$$\beta_{expanded} = k^0 + k^\delta + k^{2\delta} + \dots + k^{\lfloor \frac{n}{\delta} \rfloor * \delta}$$

This is a geometric progression (Bronshtein et al. 1997) with initial term  $a=1$  and ratio  $r=k^\delta$ . Hence, the sum is given by

$$\beta_{expanded} = \frac{a(r^{n+1}-1)}{r-1} = \frac{k^{(\lfloor \frac{n}{\delta} \rfloor + 1) * \delta} - 1}{k^\delta - 1} \quad (1)$$

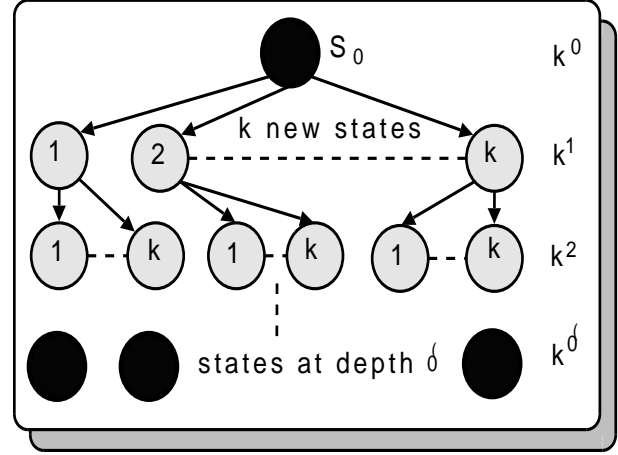


Figure 7: At depth  $d$ , the number of states is  $k^d$ . All states at depth  $\delta$  are explicit

Similarly, the total number of states is another geometric progression with initial term  $a=1$  and ratio  $r=k$ .

$$\beta_{total} = k^0 + k^1 + k^2 + \dots + k^n$$

$$\text{or } \beta_{total} = \frac{a(r^{n+1}-1)}{r-1} = \frac{k^{n+1}-1}{k-1} \quad (2)$$

Therefore the number of states in difference form is given by

$$\beta_{difference} = \beta_{total} - \beta_{expanded}$$

Assigning  $\beta_{expanded}$  from equation 1 and  $\beta_{total}$  from equation 2 we get

$$\beta_{difference} = \frac{k^{n+1}-1}{k-1} - \frac{k^{(\lfloor \frac{n}{\delta} \rfloor + 1) * \delta} - 1}{k^\delta - 1}$$

**Percentage Reduction in Memory:** The number of states in difference and explicit forms is given by equations 1 and 2. Suppose the memory occupied by an explicit state is  $\lambda$ , while a state stored in difference form occupies  $x*\lambda$  memory, where  $0 < x < 1$ . Therefore the memory needed to generate a reachability graph of depth  $n$  without using our algorithm is

$$\Lambda_{withoutalgo} = \beta_{total} * \lambda \quad (3)$$

When using our algorithm, the memory needed to generate the same reachability graph is

$$\Lambda_{withalgo} = \beta_{difference} * \lambda * x + \beta_{expanded} * \lambda \quad (4)$$

The percentage reduction in memory denoted by  $\Delta$  is

$$\Delta = \frac{\Lambda_{withoutalgo} - \Lambda_{withalgo}}{\Lambda_{withoutalgo}} \quad (5)$$

Using equations 3 and 4 in 5, we get

$$\text{or } \Delta = \frac{\beta_{total} * \lambda - \beta_{difference} * \lambda * x - \beta_{expanded} * \lambda}{\beta_{total} * \lambda}$$

Substituting  $\beta_{difference}$  as  $\beta_{total} - \beta_{expanded}$

$$\Delta = (1-x) * \left( 1 - \frac{\beta_{expanded}}{\beta_{total}} \right)$$

$$\text{or } \Delta = (1-x) * \left( 1 - \frac{(k^{(\lfloor \frac{n}{\delta} \rfloor + 1) * \delta} - 1) * (k-1)}{(k^\delta - 1) * (k^{n+1} - 1)} \right)$$

**Time needed for Extra Processing:** The extra time required when states are stored in difference form is now calculated. We only consider the delays due to backtracking as any other delay is common for both explicit and difference states.

Let  $i$  be an integer between 1 and  $n$ . If the height of reachability graph is  $i$ , the number of states in explicit and difference forms are given by

$$\beta'_{total} = \frac{k^{i+1}-1}{k-1}, \beta'_{expanded} = \frac{k^{(\lfloor \frac{i}{\delta} \rfloor + 1) * \delta} - 1}{k-1}$$

$$\text{and } \beta'_{difference} = \frac{k^{i+1}-1}{k-1} - \frac{k^{(\lfloor \frac{i}{\delta} \rfloor + 1) * \delta} - 1}{k^\delta - 1}$$

When a state  $S_{st}$  is generated, it is compared with the state stored at an index given by the hash-function. The probability that this state is stored in expanded or difference form can be calculated as

$$P_{expanded} = \frac{\beta'_{expanded}}{\beta'_{total}} \text{ and } P_{difference} = \frac{\beta'_{difference}}{\beta'_{total}}$$

If the state is stored in difference form, it has to be first expanded by backtracking and then compared with  $S_{st}$ . Hence, time taken for comparing  $S_{st}$  with the stored states is given by

$$T_{comparison} = T_{expanded} + T_{difference}$$

Suppose the time for comparing two expanded states is  $\epsilon$ , while it takes  $y * \epsilon$  time for backtracking a single step. In the worst case, a backtracking of  $(\delta - 1)$  steps is necessary to expand the state. Therefore the time can be calculated as

$$T_{comparison} = P_{expanded} * \epsilon + P_{difference} * \epsilon(\delta - 1)y$$

$$= \frac{\epsilon(1-y(\delta-1))\beta'_{expanded}}{\beta'_{total}} + \epsilon(\delta - 1)y$$

$$= \frac{\epsilon(k^{(\lfloor \frac{i}{\delta} \rfloor + 1) * \delta} - 1)(k-1)(1-y(\delta-1))}{(k^\delta - 1)(k^{i+1} - 1)} + \epsilon(\delta - 1)y$$

This is the time taken for comparing a state generated with a stored state in difference form. All comparison at a particular depth takes place concurrently. Hence the total time taken to generate reachability graph of height  $n$  is the sum of time taken for one comparison at each level. This is denoted by  $\pi$ , where

$$\pi = \sum_{i=0}^n \frac{\epsilon(k^{(\lfloor \frac{i}{\delta} \rfloor + 1) * \delta} - 1)(k-1)(1-y(\delta-1))}{(k^\delta - 1)(k^{i+1} - 1)} + \epsilon(\delta - 1)y$$

Since  $\lfloor \frac{n}{\delta} \rfloor = 0$  for  $0 \leq i < \delta$ ,  $\lfloor \frac{n}{\delta} \rfloor = 1$  for  $\delta \leq i < 2\delta$  etc.,

$$\pi = \sum_{i=0}^{\delta-1} \frac{\epsilon(k^\delta - 1)(k-1)(1-y(\delta-1))}{(k^\delta - 1)(k^{i+1} - 1)} +$$

$$\sum_{i=\delta}^{2\delta-1} \frac{\epsilon(k^{2\delta} - 1)(k-1)(1-y(\delta-1))}{(k^{2\delta} - 1)(k^{i+1} - 1)} + \dots +$$

$$\sum_{i=z\delta}^n \frac{\epsilon(k^{(z+1)\delta} - 1)(k-1)(1-y(\delta-1))}{(k^{(z+1)\delta} - 1)(k^{i+1} - 1)} + \epsilon(\delta - 1)y$$

where  $z = \lfloor \frac{n}{\delta} \rfloor$ . This is the time taken to generate a reachability graph of height  $n$  when the proposed algorithm is used.

#### 4 Experimental Result

The proposed algorithm was tested on a desktop with 2.8GHz Intel Pentium D processor and 1GB RAM. The desktop had Ubuntu 8.04 desktop version OS installed and our C source code was compiled using GNU C compiler(gcc).

We used six different Coloured Petri-net models to run our experiment. The number of places and tokens in each CPN model is listed in Table 1 and Table 2. If a model had  $m$  tokens and  $n$  places, each token was assigned an integer name  $i: i \in [0, m-1]$  and each place was assigned an integer name  $j: j \in [0, n-1]$ . Initially, all tokens were in place 0. At each state, the set of enabled transitions were selected randomly and one of

these transitions was fired. This allow having a large number of transitions in a model without specifying the bindings for which they are enabled.

For each CPN model, we have calculated the time and space needed to generate first 500 unique states using sequential algorithm and without using it. Furthermore, sequential algorithm require a non-negative integer value of  $\delta$  and we have assigned it the set of values  $\{1, 2, 3, 7, 20\}$ . When proposed algorithm is not used, we assign 0 to  $\delta$ . The results are listed in Table 1 and Table 2.  $\delta$  is the shortest distance between two explicit states.

Table 1 shows the memory needed(given by  $\Lambda$ ) to store first 500 states of CPN models used and the percentage reduction in memory requirement(given by  $\Delta$ ) for different values of  $\delta$ . For each model, the memory requirement is highest either when not using our algorithm( $\delta=0$ ), or when using it with  $\delta=1$ . In Figure 8, memory required is plotted against value of  $\delta$ . On increasing the value of  $\delta$ , the memory requirement decrease for all models. Furthermore, the decrease is significantly higher for large models, with a significantly greater number of places and tokens, as compared to small models. For instance, the CPN model with 1500 places and 2000 tokens used 95% less space when our algorithm was used with  $\delta=20$ . Compared to this, the reduction was 76% for a CPN model with 4 places and 5 tokens. Nevertheless, the reduction is massive for models of all size and for all values of  $\delta$ , as evident from Table 1 and Figure 8.

Table 2 shows the time needed(given by  $\pi$ ) to generate first 500 states of CPN models used and the percentage increase in delay(given by  $\eta$ ) for different values of  $\delta$ . For each model, the delay is minimum when our algorithm is not used( $\delta=0$ ). In Figure 9, delay is plotted against value of  $\delta$ . When our algorithm is used, delay increase with increase in value of  $\delta$ . This increase is massive for small models. A model with 4 states and 5 tokens generates 500 states almost instantly( $\pi=0$ ) without using our algorithm. The same model needs 1.5 second when our algorithm is used with  $\delta=20$ . The percentage increase in delay ( $\eta$ ) decrease with an increase in size of model. The model with 1500 places and 2000 tokens has twice the delay when our algorithm is used with  $\delta=20$  as compared to when  $\delta=0$ (algorithm not used). A comparison of results in Table 1 and Table 2 clearly shows that the reduction in memory requirement comes at the cost of extra delay in processing. This is further evident in Figure 10 where the required memory decrease and delay increase with increase in value of  $\delta$ . However, the memory reduction is massive as compared to the increase in delay, especially for moderate to large size models. A model with 1500 places and 2000 tokens had 95% reduction in memory with double the delay. Due to inherent complexity of most modern systems, their models are almost always large. Our algorithm is addressing a niche for such systems.

#### 5 Discussion

The proposed algorithm reduce the memory requirement for model checking by storing states in difference form and thereby allow model checking in a machine with a fraction of memory needed otherwise. This might lead to wider use of model checking in software verification and subsequent production of reliable software systems. Although there is an increase in delay due to backtracking, the results illustrate that the delay is small when compared to the massive reduction in memory obtained.

**Reduction in memory requirement  $\Lambda$ :** State-space analysis without using our algorithm will store all states in explicit form, leading to maximum mem-

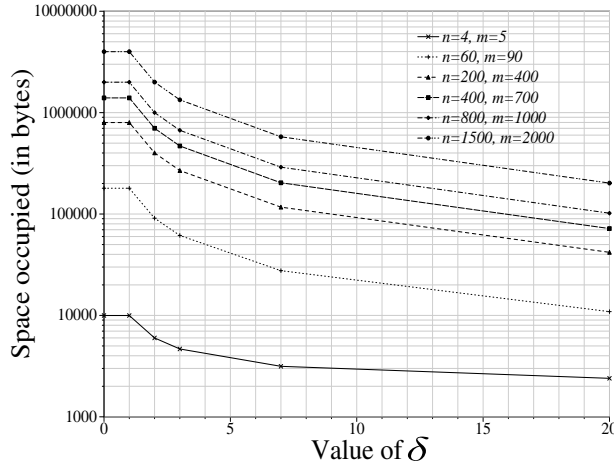
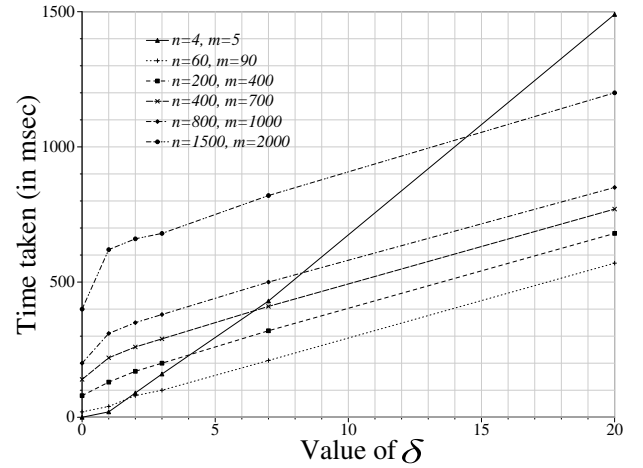
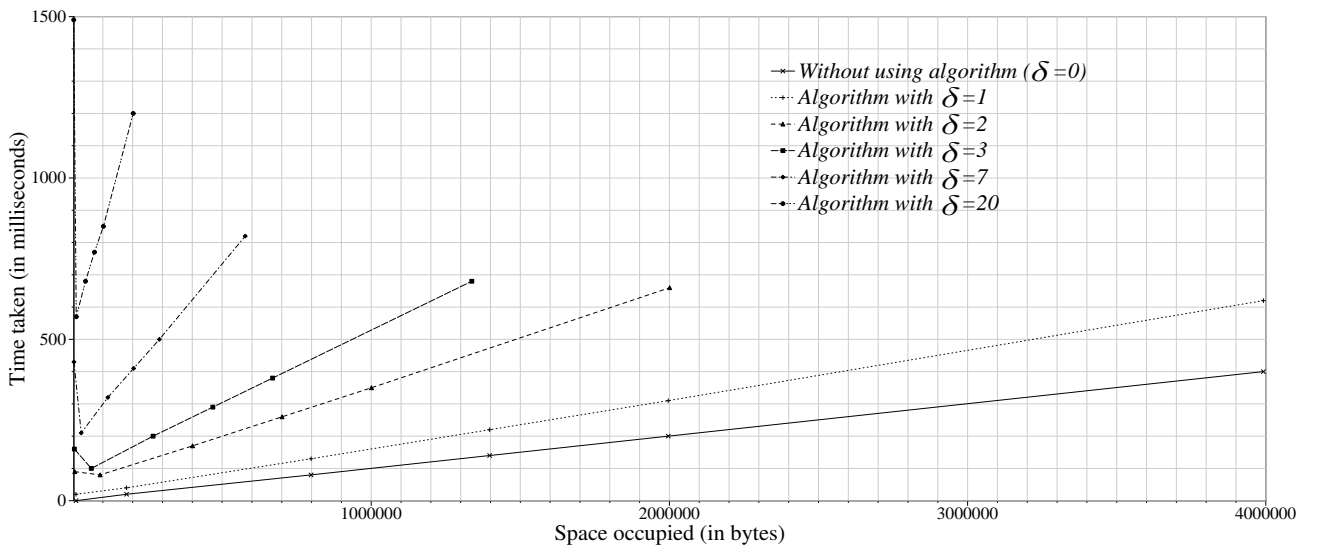


Table 1: Space occupied(in bytes) by first 500 states of CPN models( $\Lambda$ ) and percentage decrease in space( $\Delta$ )

n	m	$\delta=0$	$\delta=1$		$\delta=2$		$\delta=3$		$\delta=7$		$\delta=20$	
		$\Lambda$	$\Lambda$	$\Delta$	$\Lambda$	$\Delta$	$\Lambda$	$\Delta$	$\Lambda$	$\Delta$	$\Lambda$	$\Delta$
4	5	9980	9980	0%	5996	40%	4668	53%	3148	68%	2396	76%
60	90	179640	179640	0%	90996	49%	61448	66%	27628	85%	10896	94%
200	400	798400	798400	0%	400996	50%	268528	66%	116908	85%	41896	95%
400	700	1397200	1397200	0%	700996	50%	468928	66%	203308	85%	71896	95%
800	1000	1996000	1996000	0%	1000996	50%	669328	66%	289708	85%	101896	95%
1500	2000	3992000	3992000	0%	2000996	50%	1337328	67%	577708	86%	201896	95%

Table 2: Time(in msec) to generate first 500 states of CPN models( $\pi$ ) and percentage increase in time( $\eta$ )

n	m	$\delta=0$	$\delta=1$		$\delta=2$		$\delta=3$		$\delta=7$		$\delta=20$	
		$\pi$	$\pi$	$\eta$	$\pi$	$\eta$	$\pi$	$\eta$	$\pi$	$\eta$	$\pi$	$\eta$
4	5	$\approx 0$	20	$\infty$	90	$\infty$	160	$\infty$	430	$\infty$	1490	$\infty$
60	90	20	40	100%	80	300%	100	400%	210	950%	570	2750%
200	400	80	130	62%	170	112%	200	150%	320	300%	680	750%
400	700	140	220	57%	260	85%	290	107%	410	193%	770	450%
800	1000	200	310	55%	350	75%	380	90%	500	150%	850	325%
1500	2000	400	620	55%	660	65%	680	70%	820	105%	1200	200%

Figure 8: Memory requirement decrease with increase in value of  $\delta$ .  $\delta=0$  means algorithm not used. Y-axis uses log scale.Figure 9: Delay increase with increase in value of  $\delta$ .  $\delta=0$  when algorithm not used.Figure 10: Delay increase and memory requirement decrease with increase in value of  $\delta$ . Each curve is for a different value of  $\delta$  as indicated by the legend. The six points on a curve correspond to six CPN models used

ory requirement. This holds for our results in Figure 8. Furthermore, using our algorithm with  $\delta=1$  also stores all states in explicit form, keeping  $\Lambda$  unchanged. However when  $\delta=2$ , every alternate state is stored in explicit form. This leads to almost 50% reduction in  $\Lambda$  as only half the total number of states are in explicit form. Similarly, when  $\delta=3$ , one in every three states are stored in explicit form leading to 66% reduction in  $\Lambda$ . When  $\delta=7$ , one in seven states is stored in explicit form leading to 85% reduction in  $\Lambda$ . Finally, when  $\delta=20$ , one in 20 states is stored in explicit form resulting in 95% reduction in value of  $\Lambda$ .

The reduction for CPN model with 4 places and 5 tokens is low as compared to other models. The reason being that the size of an explicit state is almost same as a difference state for a small model. Therefore, replacing explicit state with difference state do not make a big difference.

**Increase in delay  $\pi$ :** Two factors contribute to overall delay: 1)backtracking to expand a difference state 2)when state-space is being explored using DFS algorithm and a duplicate state is encountered, the stack is popped till a state with an enabled event(transition) is encountered. Popping a stack is time intensive operation.

A small model has less number of possible states and therefore the chances of encountering a duplicate state is high. The CPN model with 4 places and 5 tokens encountered 469 duplicate states before generating 500<sup>th</sup> unique state. As compared to this, the model with 60 places and 90 tokens encountered only 1 duplicate state before generating 500<sup>th</sup> state. The delay in popping stack, combined with backtracking delay lead to large  $\pi$  for small models.

When model-checking, we need not backtrack if all states are in explicit form. This leads to low  $\pi$  when  $\delta=0$  or  $\delta=1$ . However, due to extra processing delay of our algorithm, the delay for  $\delta=1$  is higher than  $\delta=0$ . On further increasing  $\delta$ , delay increases due to backtracking. Higher the value of  $\delta$ , more is the backtracking needed to expand a state and greater the delay.

## 6 Related Work

All solutions proposed to store state-space can be classified as either of 1)Exhaustive storage 2)Partial storage or 3)Lossy storage. The proposed algorithm is based on exhaustive storage, wherein all explored states of a model are compressed and stored in a suitable data structure(e.g. hash-table) to ensure constant time lookup. Table 3 compares proposed algo-

Table 3: A comparison of solutions based on exhaustive storage

Method	Run-Time	Memory-Use
No Algorithm	100%	100%
(Schmidt 2003)	130%	60%
(Evangelista & Pradat-Peyre 2005)	300%	05%
(Holzmann 1997)	280%	18.3%
Sequential Algorithm proposed	200%	05%

rithm with other solutions based on this approach and the state-space compression they provide. The table also gives the additional delay incurred when using a solution. The proposed algorithm provides reduction equivalent to (Evangelista & Pradat-Peyre 2005) with only 2/3 of its delay.

In Partial storage, only a subset of the explored states are stored. Sweep-line method, proposed in

(Christensen et al. 2001), is a solution based on partial storage where a state is deleted if it cannot be reached again in future. However, it is difficult to decide the states to be deleted. Furthermore, it is not a generic solution as for different systems, we might need to delete a different set of states.

Lossy storage is similar to exhaustive storage wherein explored states are stored in compressed form in suitable data structure. However, it is not possible to decompress the states. In order to determine if a state is new, it is also compressed and compared with stored states. As pointed out previously, multiple states can have same compressed form. This often results in falsely implicating a state as duplicate. An interesting solution based on lossy storage is proposed in (Wolper et al. 1993)

## 7 Conclusion

In this paper, we have reduced the memory requirement for model checking by storing states in difference form. Consequently, model checking would acquire a bigger role in verification of a wide range of softwares. This will ensure safety and reliability of software systems used in all walks of life. Experimental results indicate that our algorithm performs remarkably better for large models. Contemporary systems have high level of complexity, often leading to large models. The proposed algorithm is addressing a niche for such systems.

In future, we aim propose a distributed model checking algorithm by extending the sequential algorithm. This will reduce the accompanying delay as a result of parallel processing.

## References

- Basic Spin Manual* (2007).  
**URL:** <http://spinroot.com/spin/Man/Manual.html>  
 Beizer, B. (1990), *Software testing techniques*.  
*Blast Manual* (2008).  
**URL:** <http://www.cs.sfu.ca/~dbeyer/blast/doc/blast.pdf>  
 Bronshtein, I. N., Semendyayev, K. A. & Kirsch, K. A. (1997), *Handbook of mathematics (3rd ed.)*, Springer-Verlag, London, UK.  
 Christensen, S., Kristensen, L. M. & Mailund, T. (2001), A sweep-line method for state space exploration, in 'TACAS 2001: Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems', pp. 450-464.  
 Clarke, E., Grumberg, O. & Peled, D. (2000), *Model Checking*, MIT Press.  
 Clarke, E. M. & Berezin, S. (1998), 'Model checking: Historical perspective and example (extended abstract)'.  
 Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2001), *Introduction to Algorithms*, 2nd revised edition edn, The MIT Press.  
*CPN Tools* (2009).  
**URL:** <http://wiki.daimi.au.dk/cpntools/cpntools.wiki>  
 Evangelista, S. & Pradat-Peyre, J.-F. (2005), Memory efficient state space storage in explicit software model checking, in 'SPIN Workshop on Model Checking of Software', pp. 43-57.  
 Holzmann, G. J. (1997), State compression in spin: Recursive indexing and compression training runs, in 'In Proceedings of Third International SPIN Workshop'.  
 Jensen, K. (1997), *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts*, Springer-Verlag.  
 Rushby, J. (1989), Formal methods and critical systems in the real world, in 'Formal Methods for Trustworthy Computer Systems (FM89)', pp. 121-125.  
 Schmidt, K. (2003), Using petri net invariants in state space construction., in 'Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003), 9th International Conference', Springer Verlag, pp. 473-488.  
 Wolper, P., , Wolper, P. & Leroy, D. (1993), Reliable hashing without collision detection, in 'In Computer Aided Verification. 5th International Conference', Springer-Verlag, pp. 59-70.

# Efficient Contour Line Labelling for Terrain Modelling

Xin Xie

Burkhard C. Wünsche

Graphics Group, Department of Computer Science  
University of Auckland,

Private Bag 92019, Auckland 1142, New Zealand,

Email: xxie021@aucklanduni.ac.nz, burkhard@cs.auckland.ac.nz

## Abstract

Terrains are an essential part of outdoor environments. Terrain models are important for computer games and applications in architecture, urban design and archaeology. A popular and intuitive way to represent terrains is by contour maps. In order to render such representations in 3D the contours must be labelled with height values and converted to Digital Elevation Maps (DEM), which are regular grids of height values and are represented as gray scale images.

The labelling of contour lines is time intensive for large maps and prone to errors. In this paper we present an efficient and novel algorithm for semi-automatically labelling contour maps and for converting them to DEMs. The algorithm first identifies *point extrema* which must be labelled by the user. The point extrema are connected by a graph and the contour lines crossed by edges are labelled automatically. We show that ambiguities can exist for so-called *line extrema*. Our algorithms will resolve ambiguous regions requiring a minimal number of additional user inputs. We also present a more efficient graph representation, which requires about 10-20% more user inputs than the optimal case. After the contour lines are labelled, the contour map is triangulated and the height value at each point of the DEM is computed using a bilinear interpolation.

The presented algorithm is efficient, requires minimal user inputs, and produces good quality DEMs. We present several examples, discuss its suitability for different applications, and provide a complexity analysis.

**Keywords:** contour map, digital elevation map, shortest-path spanning tree, Delaunay triangulation, terrain visualization

## 1 Introduction

Terrain modelling is fundamental to a wide range of computer applications, such as games, flight simulators, urban design and planning, archaeology, and geography and engineering visualizations. The two most popular representations of terrains are contour maps and Digital Elevation Maps (DEM).

Digital elevation maps consist of a regular grid of height values and can be stored and represented by gray scale images where the gray level of each pixel represents the height value at that location. White/black represent the highest/lowest altitude,

respectively. DEMs are popular for rendering since they can be compressed, are suitable for out-of-core rendering, and can be converted to progressive and multi-resolution representations, which are essential for real-time rendering of massive data sets (Duchaineau et al. 1997, de Boer 2000, Losasso & Hoppe 2004). Human users often have difficulties to perceive terrain details from DEMs since the human visual system's gray scale resolution is limited and colour/gray scale perception is non-linear. The ability to compare colours (gray scales) at different locations is even more limited since the perceived colour depends on the colour of surrounding regions (simultaneous contrast) (Schiffman 1996). Many of these disadvantages can be resolved by rendering the DEM in 3D. This however, requires complex rendering algorithms, more screen space and animations to fully perceive the 3D geometry.

A more effective and intuitive representation of terrains for human viewers are contour maps, also known as topographic maps. Contour maps consist of a series of non-intersecting curves or lines, so-called contour lines, which represent points of equal height in the terrain. Contour lines are always one-dimensional, i.e., they cannot represent planar regions of constant height, and are either closed or have endpoints on the boundary of the image. Contour maps are usually prepared using airborne and satellite altitude measurements, or photogrammetric interpretation of aerial photography (Wikipedia n.d.), which gives an accurate depiction of terrain features (U. S. Army Corps of Engineers and American Society of Civil Engineers 1996). In contrast to DEMs contour maps allow height comparisons of spatially distinct or neighbouring regions. Magnitude and direction of gradients can be perceived from the contour line density and orientation. Figure 1 shows an example of a contour map and the corresponding digital elevation map.

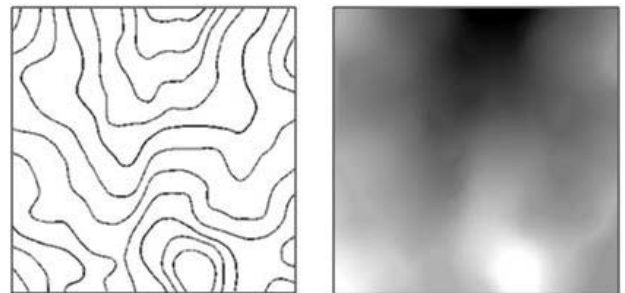


Figure 1: Example of a contour map (left) and the corresponding digital elevation map (right).

In order to display terrains represented by 2D contour maps in 3D they must be converted to DEMs. A precondition for this conversion is that height values

are given for all contour lines. This is usually the case when using Geographic Information Systems (GIS), but for many cases such height values must be input by hand which is cumbersome and prone to errors. Our research was motivated by two applications: one application was an archaeological visualization system where we were given hand drawn contour maps of a terrain (Kymer 2009). The second application is a sketched based terrain modelling tool, where users create 3D terrains by sketching contour lines, mountain peaks and rivers. Examples of both applications are used in the result section.

Section 2 reviews previous work on generating and converting contour maps. Section 3 and 4 present definitions, properties and theorems required for the development of an efficient algorithm for labelling contour lines. The resulting algorithm and implementation details are presented in section 5 and 6, respectively. Results are given in section 7 and are followed by our conclusions and suggestions for future work.

## 2 Literature Review

Contour maps are the most common representation of terrains in maps and the 3D visualization of map data improves realism and recognition of terrain features. Consequently there have been many projects concerned with the conversion of contour maps to DEMs. The majority of research concentrates on the image processing of map data and the interpolation of contour line data.

Image processing research is concerned with the recognition of contour lines from digitised map data, especially in the presence of textured backgrounds and noisy data. A typical procedure consists of the following four steps (Arrighi & Soille 1999):

1. digitalization of the topographic map, e.g., by using a scanner
2. thresholding
3. thinning of the black patterns by using some skeletonization procedure and
4. raster-to-vector conversion of the resulting thinned lines.

Arrighi and Soille extend this procedure based on advances in morphological image processing (Arrighi & Soille 1999). The most attractive aspect of their method is the reconnection of contour lines that are disconnected by coordinate grid lines or inscriptions. The authors extract the two ends of a line before skeletonising them by generalizing the hit-to-miss transform. Rivas and de La Fraga use a similar method, but also discuss an improved way to generate sample points from contour lines in order to create smoother terrain meshes (Rivas & de la Fraga 2005). Lalonde and Li perform contour line extraction with the use of colour information. They extract the basic colours of an image by switching from RGB to  $L^*a^*b$  colour spaces, projecting the image on its principal axes and using modified histogram splitting (Lalonde & Li 1997). A smoothness function assists with the reconnection of broken contour lines.

Once contour lines have been identified they must be labelled. When reading map data this can be achieved using character recognition techniques and by finding the corresponding contour lines, usually by using a distance metric criteria. Surprisingly we found only one previous research project concerned with our research, the semiautomatic assignment of height values to contour lines. Maia and Xavier (Maia & Átila L. F. Xavier 1996) construct a nested tree by representing each contour line by a tree node. The tree reflects the relationship among all contours in the

domain. Each node is then given an ambiguity interval which is initialised to  $[-\infty, +\infty]$  or a correct elevation value where possible. If a true elevation value is known the information is propagated through the tree in order to reduce each node's ambiguity interval. The authors claim that this method can effectively eliminate human interactions in the elevation assignment process such that human errors remain at the minimum.

The last step necessary for computing DEMs is to compute height values at its regular grid points. This requires interpolation of height values at irregular sample points, usually obtained by sampling the contour lines. Dakowicz and Gold introduce three interpolation methods based on a Delaunay triangulation of the terrain (Dakowicz & Gold 2002): bilinear (barycentric) interpolation of triangle vertex values, and a so-called "Gravity interpolation" and "Sibson interpolation". The central idea of the "Gravity interpolation" is that the weighting of each data point used is inversely proportional to the square of the distance from the data point to the grid node being estimated, whereas the "Sibson interpolation" inserts each grid point temporarily into the Voronoi diagram of the data points and measures the area taken from each of a well-defined set of neighbours. Rognant et al. present a dual representation of contour lines and DEMs based on a Constrained Delaunay Triangulation (CDT) (Rognant et al. 2001). The data structure might be useful in interactive modelling applications using both contour and DEMs representations.

## 3 Definitions and Properties

### 3.1 Properties of Contour Lines

We did not find any precise definition of properties of contour lines. In many cases the actual properties depend on how the contour map was derived. We developed a set of properties which conforms to (often implicitly assumed!) properties used in the literature and with the properties of contour lines in widely used GIS software packages such as ArcInfo:

**Equidistance:** The height difference  $\Delta_{height}$  between two neighbouring lines is constant (or zero if lines belong to the same iso-level).

**Completeness:** Contour lines contain no gaps and they are either closed or end at the map boundaries.

**Orthogonality:** The contour lines are orthogonal to the terrain gradient and the terrain gradient is orthogonal to the contour line. This also means that the contour lines must be smooth, i.e., contain no corners.

**Continuously differentiable:** The contour lines do not overlap, intersect or branch. The terrain gradient can be extracted from the distance and height difference between points on the contours.

Virtually all contour maps we examined fulfilled these properties. Exceptions were digitised contour maps where gaps could occur due to sampling errors and mathematically defined contour maps where branching can occur, e.g., for a function with a cross shaped extrema at an iso-level. Such cases are virtually non-existence for real terrains and would not be produced by software generating contour data.

### 3.2 Extrema in Contour Maps

A contour map can have four topographical types of extrema:

1. Closed contours, which have no other contours within it (e.g., mountain tops)
2. Contours connected to the boundary, which have no other contours on one side (e.g., slopes intersecting the boundary)
3. Closed contours containing additional closed contours and being higher/lower than the adjacent contours (e.g., the rim of a volcanic crater)
4. Contours connected to the boundary, with the contours on both sides being higher/lower (e.g., valleys/ridges)

For the purpose of our research we can differentiate these four topographical types of extrema into two topological types:

**Definition 1** A Point Extremum is a contour line which encloses, possibly together with part of the boundary, a region containing either no contours or all other contours.

This definition includes case (1) and (2) of the topographical types of extrema. Note that all points in the region enclosed by a point extremum are considered to have identical height values. A point extremum can therefore be reduced to a single point without changing the topology of the terrain - hence the choice of the name. Note that if a contour line does not have neighbouring contour lines on both sides, then it is a point extremum. Several examples of point extrema are shown in red in the two images on the left hand side of figure 2.

Point extrema can be visually identified as closed contours or contours connected to the boundary, which contain no other contours. Section 6 will present an efficient algorithm for identifying point extrema which uses a Delaunay triangulation required for interpolating height values.

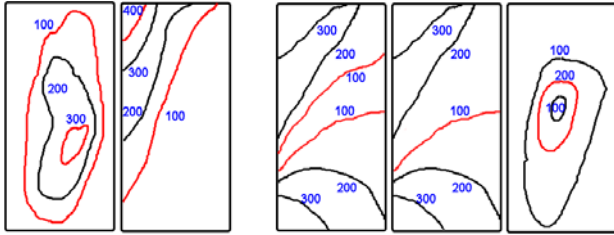


Figure 2: Left: Two images showing point extrema (red). Right: Three images showing line extrema (red).

**Definition 2** A Line Extremum is a contour line which is either not higher or not lower than the contour lines on either side of it.

Several examples of line extrema are shown in red in the three images on the right hand side of figure 2. In contrast to point extrema, line extrema cannot be detected from the contour information alone. As an example consider a domain with three contours having the elevations:

**Case 1:** 400m, 300m and 200m

**Case 2:** 400m, 300m and 400m

In either case the contour lines on the left and right can be identified as point extrema. However, only in the second case the contour line in the middle is a line extremum.

Note that the line extremum in case (2) can be identified by providing height values for the point extrema on either side. However, even in this case the

height value of the contour line in the middle cannot be determined since it could be lower (300m) or higher (500m) than its neighbouring contour lines. In contrast, in case (1) the height value of the centre contour line can be immediately determined from the heights of the neighbouring contours.

This observation suggests that an effective way to label all contour lines is to first label the point extrema and then deduce the height values in between automatically wherever possible. The following subsection proves several theorems required for the development of an efficient algorithm for labelling contour lines.

#### 4 Graph-Theoretical Foundations

As indicated in the previous section our proposed solution will be based on labelling point extrema first. These extrema can be easily identified and, as we proof next, always exist:

**Theorem 1** If a domain contains one or more contour lines, then it has at least one point extremum.

**Proof.** Suppose that a domain or its subset contains one or more contour lines and none of them are point extrema.

**Case 1** If the contour line is closed and has no other contours inside, then by definition it is a point extremum, which contradicts the assumption.

**Case 2** If the contour line is closed but contains one or more contours, then the contours must be closed (i.e., not connected to the boundary) since otherwise they would intersect the surrounding contour. In this case the innermost one is a point extremum (proof by induction), which contradicts the assumption.

**Case 3** If the contour line is connected to the boundary and the region surrounded by it contains no other contours, then by definition it is a point extremum, which contradicts the assumption.

**Case 4** If the contour line is connected to the boundary and the region surrounded by it contains  $N$  contours, then it either contains a closed contour (case 1 or 2), or a contour connected to the boundary forming a region with at most  $N-1$  contours (case 3 or 4). By structural induction this case can be reduced to case 1-3, i.e., the region contains a point extremum, which contradicts the assumption.

Hence by proof of contradiction any domain containing contour lines contains at least one point extremum.  $\square$

**Theorem 2** The height value of a point extremum cannot be inferred from the heights of its neighbouring contour lines.

**Proof.** By definition a point extremum is a contour line which encloses a region which contains no other contour lines. Hence it has neighbouring contour lines on at most one side. Even if the height value for the neighbouring contour line is known, the point extremum can be higher or lower than that contour line. Hence the height value cannot be inferred.  $\square$

Because of theorem 2 it is clear that any contour line labelling algorithm requires user input of height values for all point extrema. As mentioned previously

we will present in section 6 an algorithm for identifying these extrema. We will show next that knowledge of the point extrema is sufficient to identify all other contour lines:

**Theorem 3** *If all point extrema in a domain are connected by a spanning tree, then all contour lines will be crossed by at least one edge of the tree.*

**Proof.** Theorem 3 is equivalent to

*If there is a contour line which is not crossed by any edge of the spanning tree, then there must be a point extremum which is not connected to the spanning tree.*

Assume there is a contour line that is not crossed by any edge of the spanning tree and all point extrema are part of the spanning tree.

**Case 1** The not-crossed contour line is closed and contains no other or all other contours in the domain. In this case it is a point extremum by definition. This contradicts the assumption that all point extrema are part of the spanning tree.

**Case 2** The not-crossed contour line is open, i.e., its endpoints lie on the boundary of the domain and the line divides the domain into two regions. If one of these regions contains no contour lines, then the contour line is a point extremum by definition. This contradicts the assumption that all point extrema are part of the spanning tree. If both regions contain contour lines, then by theorem 1 both regions must contain at least one point extremum. Since we assumed that all point extrema are connected by a spanning tree this means that the spanning tree must contain at least one edge connecting one extremum from either side. Hence this edge must cross the contour line dividing the regions. This is a contradiction to the assumption that the contour line is not-crossed.

Hence by proof of contradiction all contour lines in a domain are crossed by the edges of a spanning tree connecting all point extrema in the domain.  $\square$

**Theorem 4** *If none of the contours between two point extrema is an extremum then all of these contours have either strictly monotonically increasing height values or strictly monotonically decreasing height values.*

**Proof.** Assumed that not all of the contour lines between the point extrema have either strictly monotonically increasing height values or strictly monotonically decreasing height values. In that case there is at least one contour line which either has a neighbouring contour line with equal height value or where the neighbouring contour lines are either both higher or both lower. By definition 2 this contour line is a line extremum (or point extremum if closed), which contradicts the requirement that there are no other extrema in between these two point extrema.  $\square$

## 5 Algorithm Design

Our algorithm requires that the contour map fulfils the properties listed in subsection 3.1. We also assume that it contains no other information such as text or geographic features. A significant amount of research has been going into image processing tools extracting contour lines from real map data, but is outside the scope of this paper (Xin et al. 2006, TERRAINMAP.COM - Digital Elevation Modeling Journal n.d., Arrighi & Soille 1999).

Our algorithm is based on the theorems proven in the previous section. In particular theorem 2 shows that any algorithm for labelling contour lines requires user inputs for all point extrema. Theorem 3 shows that in order to label all contours it is sufficient to consider contours crossed by the edges of a spanning tree connecting all point extrema. Hence our algorithm contains the following steps:

### Algorithm 1: Contour Line Labelling

**Step 1:** Identify point extrema and request appropriate height values from the user

**Step 2:** Create a *contour graph* whose nodes are point extrema and connect them by edges such that every node is connected to any other node. This means that the graph represents or contains a spanning tree and hence, with theorem 3 every contour line is intersected by at least one edge.

**Step 3:** Investigate the number of contour lines intersected by each edge. Let  $A$  and  $B$  be two nodes of the graph with heights  $h_A$  and  $h_B$  and  $n$  the number of contour lines intersected by the edge between those nodes and  $\Delta_{height}$  the height step between two contour lines.

**Step 3A:** If

$$h_B - h_A = (n - 1) * \Delta_{height} \quad (1)$$

then there is no extremum in between the nodes  $A$  and  $B$  and with theorem 4 we can label the intersected contour lines with strictly monotonically decreasing or increasing height values.

**Step 3B:** Otherwise all contour lines between the nodes are labelled as ambiguous and the user has to choose one of them and label them appropriately. The labelled contour line will become a new node of the graph. For all new edges it is checked whether

$$h_B - h_A > (n - 1) * \Delta_{height} \quad (2)$$

If yes, then the user input was invalid since it violates the equidistance condition of the contour map. If no, apply step 3A to each section of the edge subdivided by the new node.

Repeat step 3 until all edges of the contour graph have been processed.

**Step 4:** If all edges of the graph have been investigated, but not all contours have been labelled than the input contour map has an invalid topology, e.g., gaps between contour lines.

The algorithm has three critical aspects:

- The choice of the edges of the contour graph influences the efficiency and effectiveness of the labelling process. A full graph grows quadratically in size, but minimises user inputs in the sense that contour lines can be automatically labelled wherever possible. A spanning tree grows linearly, i.e., less contour lines must be investigated by the algorithm, and it still guarantees that all contour lines are labelled. However, in some cases it might not be optimal, i.e., its edges do not cover all cases where an automatic labelling is theoretically possible.
- If a shortest spanning tree is used edge weights must be defined. Possible choices are the Euclidean distance or the number of contours between nodes (point extrema). Different weighting functions result in different SSTs and hence can result in different numbers of user inputs.



- If an automatic labelling in step 3 is not possible the user must choose to label a line extrema in order to minimise user inputs. As explained in subsection 3.2 it is not possible to identify the exact location of a line extrema or to deduce its height value. The user must make this choice from additional information not available in the contour map. This could be a real map from which the contour map was derived or it could be design choices when modelling a contour map from scratch.

These issues will be discussed in more detail in section 7.

## 6 Implementation

### 6.1 Contour Line Indexation

We represent contour maps by gray scale images that only contain contour curves and, as mentioned previously, exclude other geographic data such as background details and inscriptions. In order to label contour lines they must be first identified and then indexed. The indexing makes it possible to determine whether two points belong to the same contour line. The contour lines are first skeletonised such that they form a simple 8-connected line, i.e., all pixels of the line are connected via horizontal, vertical or diagonal neighbouring pixels. We use an extension of a simple border tracing algorithm (Marita n.d.). A pixel is set to zero if it does not belong to a contour line and otherwise its gray scale value represents the index of the contour line to which it belongs. Note that if a map contains more than 256 contour lines other image formats could be used. For example, 3 byte RGB colours provide more than 16 million indices. More details of our implementation of the border tracing algorithm are given in (Xie 2008).

### 6.2 Sampling and Triangulation

In order to obtain a continuous height field for the domain we sample the contour lines and triangulate the sample points. We currently sample the contour lines at regular intervals (e.g., 10 pixels). Very thin triangles are avoided by adjusting sample points at the boundaries accordingly (Xie 2008). An improved solution would take into account contour density and curvature in order to avoid very small triangles and increase sample point density in regions of high curvature.

In the next step the sample points are triangulated. By interpolating height values at the triangle vertices this results in a continuous representation of the height field. Desirable conditions for a triangulation are:

1. The solution should be unique for a given set of points, regardless the sequence of points inserted.
2. The shapes of each resultant triangle should be equilateral or close to equilateral, if no other constraints are specified.
3. The vertices of each triangle are the nearest neighbour points, such that the perimeter of the triangle is minimum.

These conditions reduce numerical errors and gradient discontinuities in the resulting interpolation. The above conditions define the well-known and popular Delaunay triangulation. We use an implementation of the Bowyer-Watson algorithm (Arens 2002) which has a run-time of  $n \log(n)$  where  $n$  is the number of sample points.

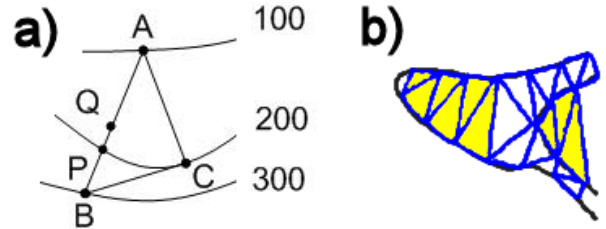


Figure 3: (a) Contour intersected by an edge of the triangulation (b) Flat area in a triangulation.

The basic Delaunay triangulation has two disadvantages. The first one is that the semantic information given by the contour lines is ignored. Figure 3 (a) shows an example: Assume the points  $A$ ,  $B$  and  $C$  are sample points and form a triangle in the Delaunay triangulation. If this triangle is then used to reconstruct height values using a bilinear (barycentric) interpolation, then for the line  $\overline{AB}$  the height value of  $200m$  would be at the point  $Q$ , whereas according to the contour lines it should be at the point  $P$ . The problem can be avoided by ensuring that the sample point distance is smaller than the contour line distance. Another solution is to force triangle edges to lie on the contour lines by using a Constraint Delaunay Triangulation (CDT) (Peterson 1998). If the sample point density is low and contour line curvature is high this can lead to triangles “cutting off” part of the contour line. In our examples we found that a standard Delaunay triangulation is sufficient. We suggest that the best solution would be a hybrid approach where constraint edges are only used in regions of high contour line density.

The second disadvantage of the Delaunay Triangulation is that it leads to unnatural flat regions as illustrated in figure 3. Note that this effect also occurs for a CDT and can even be stronger in that case. The best ways to avoid this effect are to force the triangulation to use sample points from different contours where possible or to use a higher order interpolation with a larger support (i.e., the weighting factors of the base functions also take into account height values of sample points outside the interpolated triangle).

### 6.3 Detection of Point Extrema

Subsection 3.2 showed that point extrema represent regions of equal height value. We can therefore determine point extrema from the triangulation as follows: During the triangulation give each triangle a marker “Y” if all its vertices lie on the same contour (i.e., same index) and otherwise “N”. In a second step trace all contours. If for one contour all triangles on (at least) one side have the marker “Y” then the contour is a point extrema. An example is given in figure 4. More details are found in (Xie 2008).

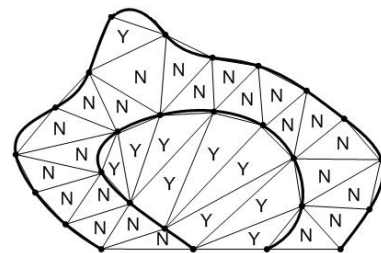


Figure 4: Examples of a point extremum (inner curve) and a normal contour line (outer curve).

## 6.4 Contour Graph

As explained in section 5 different types of contour graphs can be used for our algorithm. A *full graph* is constructed by connecting all point extrema with each other. If there are  $n$  point extrema the graph has  $n(n-1)/2$  edges. Point extrema are represented by using a random sample point on the contour. The choice of a sample point does not influence the results of the labelling process and is only visible when rendering the contour graph for illustration purposes.

The alternative is to use a spanning tree. We use a *shortest spanning tree* constructed with Prim's algorithm (Xie 2008). A more efficient implementation would be Kruskal's algorithm which is  $O(n \log n)$ . As weighting function we use the Euclidean distance between point extrema. The motivation for this is the hypothesis that the shorter the distance between two point extrema the fewer line extrema (ridges and valleys) are in-between. This would make it easier to identify line extrema.

The crossings between graph edges and contour lines are computed by drawing the graph edges as lines into an off screen buffer and checking their pixel values. If the value of a pixel is non-zero then it is intersecting a contour line and the pixel value is the contour line index. Since contour lines can have loops multiple intersections are possible. Hence new intersections are only counted if the index of the intersected contour line changes.

## 6.5 Construction of DEM

A Digital Elevation Map with  $k \times l$  pixels is constructed by computing the coordinates of each sample point with respect to the contour map and by determining for each sample point the triangle in which it lies. In order to obtain a smooth 3D rendering of the terrain it is recommended to make the sample point distance smaller than the average triangle size. Hence we loop through the triangles and determine for each triangle which sample point it contains. This can be computed in constant time since the sample points form a grid of lines parallel to the coordinate system: From the axis-aligned bounding box (AABB) and the sample distance we compute the sample points lying inside the AABB and then perform a 2D inside-outside test with the triangle edges. For each sample point we then compute a height value by performing a bilinear (barycentric) interpolation of the triangle vertex heights. Efficient formulas for this calculation can be derived by solving a linear system of equations (Xie 2008).

In order to render the DEM with common graphics and game engines we store it as a gray scale image by scaling the height values to the range  $[0, 255]$ . A smoother interpolation can be obtained by applying a smoothing filter to the image. Note that in this case the original contour height values would be lost and extrema would be flattened. If both precision and smoothness is important higher-order interpolation functions such as thin-plate splines and Catmull-Rom splines can be employed (Gousie & Franklin 1998).

## 7 Results

### 7.1 Examples

#### 7.1.1 Simple Contour Map Sketched from a Real Terrain

Figure 5 (a) shows a simple contour map representing a small section of a real terrain. The contour map was sketched by architects to represent the terrain of Selinus, an extinct city founded by the ancient Greek

civilisation in the southwest of Sicily. It has become the subject of considerable archaeological attention since the 19th century due to its remarkable architectural and artistic features, especially those of the Doric temples on the site (De Angelis 2003). The height labels have been added for illustration. The map has 7 point extrema, no line extrema and 21 contour lines.

Part (b) of the figure shows the results after sampling the contour lines and triangulating the sample points. Part (c) and (d) of the figure show a full graph and the shortest spanning tree connecting the point extrema. When using the full graph all contour lines can be labelled automatically after inputting the heights of the point extrema. When using the shortest spanning tree an ambiguity exist as illustrated in part (e) of the figure - one additional user input is necessary to resolve it. The ambiguity is caused because the contours along the bottom left edge of the SST form a "local valley". The valley does not continue over the entire map and hence the ambiguity can be resolved when using the full contour graph, but cannot be resolved when using the SST. The resulting DEM is shown in part (f) and its 3D visualization using two different view points is displayed in part (g) and (h) of figure 5.

#### 7.1.2 Example 2 - Sketched Contour Map of an Imaginary Terrain

Figure 6 (a) shows a sketched contour map which was created using free hand drawings without any terrain or map information as model. Finding appropriate height values for such free hand drawings is not trivial since they must fulfil the requirements listed in subsection 3.1. Part (b) of the figure shows the height values of all point and line extrema in red colour. The contour map has 15 point extrema, 6 line extrema, and 59 contours. Note that in the centre of the contour map there are three neighbouring contour lines with a height of 300.

Figure 6 (c) and (d) show the full contour graph (105 edges) and SST (14 edges), respectively. When using the full graph the entire map is labelled using 21 user inputs (for the 15 point and 6 line extrema). When using the SST two additional user inputs are necessary which are indicated in part (a) by yellow height values. Part (e) of the figure shows the algorithm after most of the point extrema have been labelled (shown in red). The blue lines indicate not yet labelled contours. The resulting DEM is shown in part (f) and its 3D visualization using two different view points is displayed in part (g) and (h) of figure 6.

#### 7.1.3 Example 3 - Topographic Map Data

Figure 7 (a) shows a section of a topographic map. The map was scanned in and contour lines were extracted using various image processing operators provided by the program PaintShop. Because of noise the resulting map had about 20 gaps in the contour lines which had to be fixed by hand. More advanced techniques for extracting contour lines are described in the literature (see section 2). The contour height values in part (a) of the figure were obtained by converting the contour labels in the original map from feet to meter and scaling them in order to simplify labelling and user input. Since the DEM is scaled for rendering these changes have no effect on the resulting visualization. The contour map has 24 point extrema, no line extrema, and 72 contours.

Figure 7 (c) and (d) show the full contour graph (276 edges) and SST (23 edges), respectively. When using the full graph the entire map is labelled using 24 user inputs. When using the SST six additional user



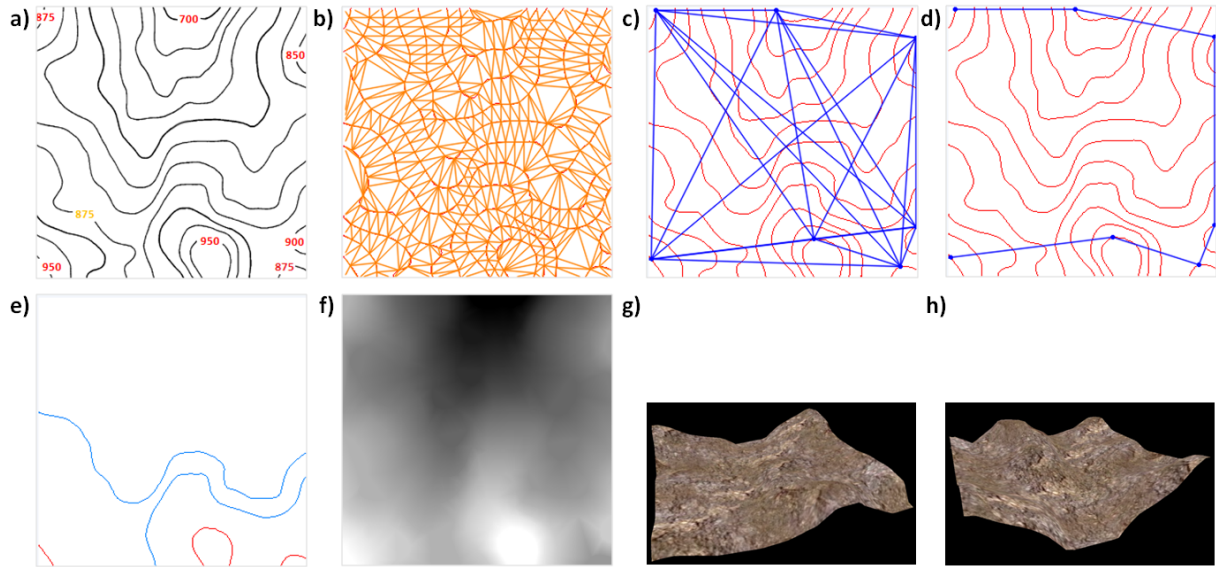


Figure 5: A sketched contour map of an archaeological site with height values (a) and the resulting Delaunay triangulation (b), full contour graph (c) and SST (d). Part (e) demonstrates that labelling using the SST results in ambiguous contours requiring an extra user input. A 3D visualization of the resulting DEM (f) using two different view points is shown in (g) and (h).

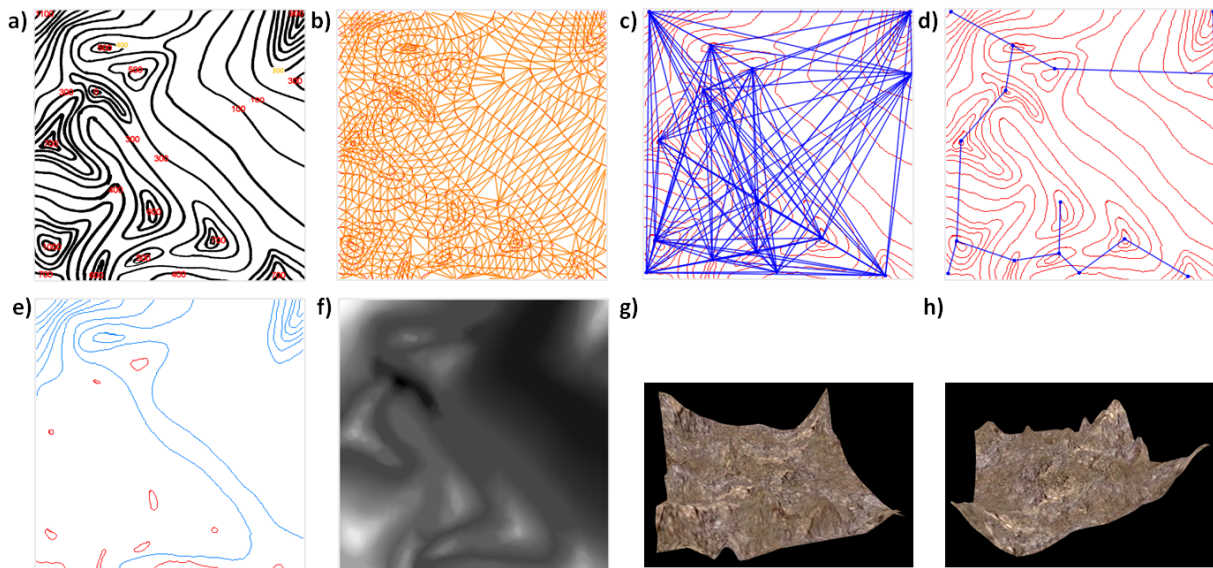


Figure 6: A sketched contour map of an imaginary terrain with height values (a) and the resulting Delaunay triangulation (b), full contour graph (c) and SST (d). Part (e) of the figure shows the algorithm after most of the point extrema have been labelled (red). The blue lines indicate not yet labelled contours. A 3D visualization of the resulting DEM (f) using two different view points is shown in (g) and (h).

inputs are necessary which are indicated in part (a) by yellow height values. Part (e) of the figure shows the algorithm after most of the point extrema have been labelled (shown in red). The blue lines indicate not yet labelled contours. The resulting DEM is shown in part (f) and its 3D visualization using two different view points is displayed in part (g) and (h) of figure 7.

## 7.2 Applications

The presented algorithm is useful for all applications where a contour map must be converted into a DEM and where the contour labels are not available in electronic form. We found three applications which are common in practice.

### 7.2.1 Sketching a Real Terrain

The first application is sketching a real terrain using contour lines as demonstrated in the first example above. While in practice many real terrains are available as GIS data, there are cases where handmade sketches are more practical and useful. Examples are terrains which have changed over time, e.g., an archaeological site which is thousands of years old, or applications where the user wants to simplify or modify the actual terrain, e.g., by adding new terrain features such as excavations representing roads or open pit mining.

Preliminary tests suggest that the interface is very intuitive if consistent height values for the overall terrain shape are available and the user only adds, deletes and modifies contours in order to model the desired landscape features. However, problems exist if no height values are given. Many users find it diffi-

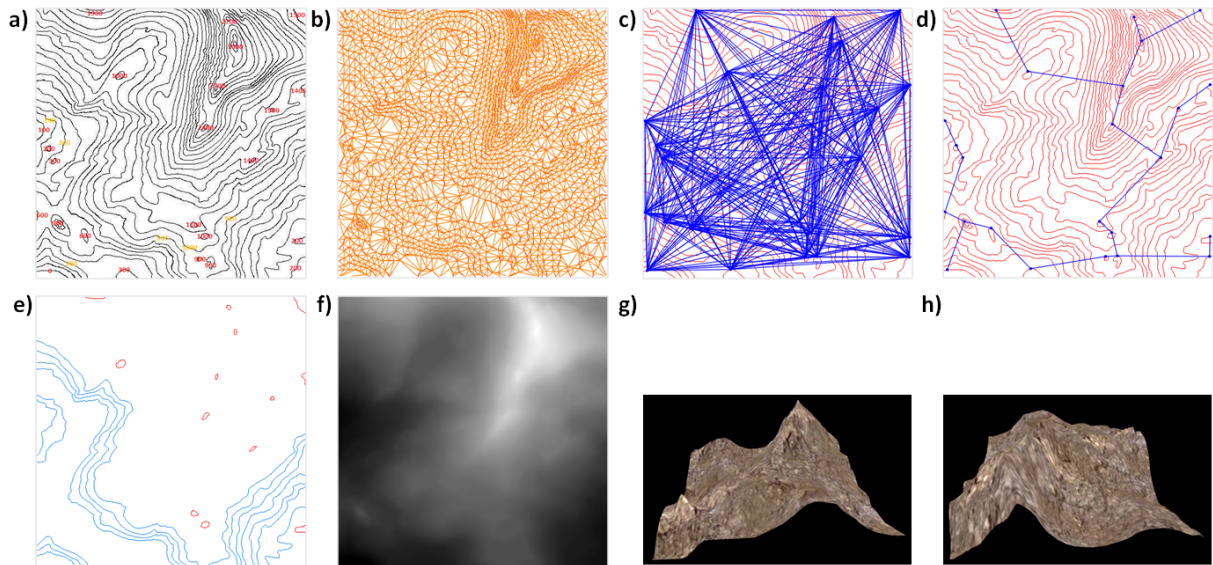


Figure 7: Section of a topographic map of a mountainous terrain. The height values have been converted from feet to meter and scaled in order to simplify labelling and user input (a). The remaining images show resulting Delaunay triangulation (b), full contour graph (c) and SST (d). Part (e) Performs the labelling using the SST results in ambiguous contours requiring an extra user input. A 3D visualization of the resulting DEM (f) using two different view points is shown in (g) and (h).

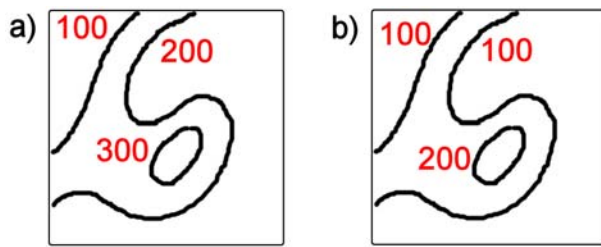


Figure 8: (a) Invalid contour labelling resulting from considering only local features. (b) Valid labelling.

cult to determine consistent and correct height labels when using a single view of a 3D terrain.

This observation resulted from a simple user study with 6 male and 3 female participants with various levels of computer graphics experience. We showed the participants the 3D terrains from figure 5 and 6 and asked them to label the corresponding contour map. All of the nine users managed to recreate the peaks of the terrain, but often didn't notice valleys and had problems estimating relative height values correctly. This was largely due to the lack of perception of terrain features rather than the inability to understand the contour label process.

Several users created impossible sequences of height labels. The main cause were consecutive contours with the same height value. The users assigned different height values which had the effect that in other parts of the map two neighbouring contours had double the allowed height difference between them. The problem is illustrated in figure 8. The results confirm the importance of the consistency test in equation 2.

Another interesting observation was that most users performed the labelling by proceeding from the lowest to the highest contour (or vice versa). This was rather surprising since the silhouettes provided a much better indication of relative height values and hence we would have expected users to first label the contours intersecting the boundary of the map. The results suggest that in applications where the point extrema are unknown a more incremental approach

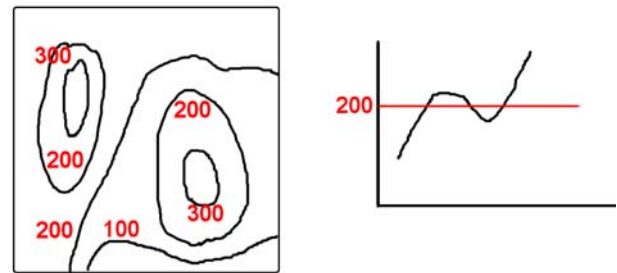


Figure 9: The labelling on the left represents an impossible configuration. The cross section on the right demonstrates that a terrain rising from 100 to 300 must intersect the 200 contour an odd number of times.

might be preferable.

### 7.2.2 Sketching an Imaginary Terrain

The second application is the sketching of an imaginary terrain as demonstrated in figure 6. While the interface is very easy to use, problems exist with the labelling of contour lines. Without any guidelines finding a consistent labelling. As in part (a) of the figure is challenging. If the point extrema are labelled first then frequently there are either too few or too many contour lines between them. While in the latter case the user can create line extrema this can lead to impossible configurations as described above. The problem is illustrated in figure 9.

Similar to the previously discussed application our experiments suggest that it is more intuitive to perform the labelling incremental. This means each time the user inputs the height value of an extremum the application performs the consistency check (equation 2) of algorithm 1 for all edges of the contour graph. This, however, is not very efficient. We are currently implementing an improved interface where the user indicates the terrain gradient by sketching arrows. Using this interface one single sketch can label all contour lines between two point extrema or line extrema. The number of necessary sketches depends

on the number of gradient changes, i.e., is equivalent to the number of edges of a subtree of the full graph connecting the extrema in the terrain.

### 7.2.3 Converting Topographic Map Data

The third application is the labelling of contour lines of a topographic map as demonstrated in figure 7. If the original map is consistent then a consistent labelling is always achieved as long as the correct height values for the extrema are input. Since the program automatically identifies point extrema the user only has to match their positions to the map and input the corresponding height values. Labelling line extrema is potentially more difficult since the program might display several potential line extrema as indicated in part (e) of the figure. If the user chooses a contour which is not a line extrema then one additional input will be necessary. The biggest problem is that the user might read the wrong height value from the topographic map in which case the resulting contour map will be inconsistent.

## 7.3 Complexity Analysis

Let the input contour image be of size  $N \times N$ . Each individual contour line has  $O(N)$  pixels. Preliminary studies performed by us suggest that most contour images use screen space efficiently, i.e., the height steps are defined such that the contour lines are neither too sparse nor too close. That means that the number  $C$  of contour lines lies between  $O(1)$  (e.g., if using a constant height step independent of image size) and  $O(N)$  (e.g., height step is adjusted such that the distance between contour lines is constant independent of the image size). In practice we hypothesize that the dimension of a contour map is similar to a fractal, i.e.,  $O(N^k)$  with  $1 \leq k \leq 2$ . More studies are necessary to verify this hypothesis.

The contour tracing, sampling and number of sample points is constant in the number of contour pixels, i.e.,  $O(N^k)$ . The Delaunay triangulation is  $O(m \log m)$  in the number of sample points  $m$ , i.e.,  $O(N^k \log N)$ .

Let  $n$  and  $e$  be the number of nodes (point extrema) and number of edges of the contour graph. For a full graph  $e = O(n^2)$  and for a SST  $e = O(n)$  where the computation of the SSP can be achieved in  $O(n \log n)$  (Kruskal's algorithm). The number of point extrema  $n$  is bounded by the number of contour lines and the average number of pixels of an edge is  $O(N)$  for a full graph, but might be smaller for a SST. Consequently an upper bound for the labelling step is  $O(N^{(k+1)})$  for a full contour graph and  $O(N^k \log N)$  if a SST created with Kruskal's algorithm is used. In the latter case the complexity of the total algorithm is bounded by  $O(N^k \log N)$  where  $1 \leq k \leq 2$  depends on the number of contour lines.

## 8 Conclusion

We have presented an efficient algorithm based on a graph-theoretical approach for contour line labelling and for converting contour maps into digital elevation maps. The algorithm automatically determines point extrema and uses them to create a contour graph. When using a full graph the number of required user inputs is optimal, i.e., the user only has to input height values for the extrema. All other contour lines are labelled automatically. When using a shortest spanning tree additional user inputs might be required, which in our tests were always around 10%–20% higher than the minimum number of user inputs.

The advantage of using an SST contour graph is that the total algorithm complexity can be bounded by  $O(N^k \log N)$  where  $1 \leq k \leq 2$  depends on the number of contour lines, which can vary between  $O(1)$  and  $O(N)$ .

The labelling algorithm can be used for contour maps obtained from digitized maps or for sketch-based terrain modelling. In the latter case it is often difficult to find consistent extrema height values and an incremental approach might be more appropriate. We have presented a simple test which the algorithm uses to verify the consistency of the contour labelling.

In future work we want to get more insight into the complexity of the algorithm, research alternative representations for the contour graph, and adapt the algorithm to make it more suitable for interactive sketch-based terrain modelling. In particular we are currently working on an application where relative heights of contours are indicated by local gradients (arrows) and classified contours have an immediate effect on the 3D terrain rendering.

## References

- Arens, C. A. (2002), The bowyer-watson algorithm: An efficient implementation in a database environment, Technical report, Department of Geodesy, Section GIS Technology, Delft University of Technology, Netherlands. URL: [http://www.gdmc.nl/publications/2002/Bowyer\\_Watson\\_algorithm.pdf](http://www.gdmc.nl/publications/2002/Bowyer_Watson_algorithm.pdf).
- Arrighi, P. & Soille, P. (1999), From scanned topographic maps to digital elevation models, in D. Jongmans, E. Pirard & P. Trefois, eds, 'Proceedings of Geovision'99: International Symposium on Imaging Applications in Geology', pp. 1–4.
- Dakowicz, M. & Gold, C. M. (2002), Visualizing terrain models from contours - plausible ridge, valley and slope estimation, in 'Proceedings of the International Workshop on Visualization and Animation of Landscape', Kunming, China.
- De Angelis, F. (2003), *Megara Hyblaia and Selinous: two Greek city-states in archaic Sicily*, University School of Archaeology monographs, 55, Oxford University Press, New York.
- de Boer, W. H. (2000), 'Fast terrain rendering using geometrical mipmapping'. URL: [http://www.flipcode.com/archives/article\\_geomipmap.pdf](http://www.flipcode.com/archives/article_geomipmap.pdf).
- Duchaineau, M., Wolinsky, M., Sigeti, D. E., Miller, M. C., Aldrich, C. & Mineev-Weinstein, M. B. (1997), ROAMing terrain: Real-time optimally adapting meshes, in 'Proceedings of Visualization '97', pp. 81–88.
- Gousie, M. & Franklin, R. (1998), Converting elevation contours to a grid, in 'Proceedings of the 8th International Symposium on Spatial Data Handling', pp. 647–656. URL: <http://citeseer.ist.psu.edu/article/gousie98converting.html>.
- Kymer, D. J. (2009), User interfaces for the effective exploration and presentation of virtual archaeological sites, Master's thesis, Department of Computer Science, University of Auckland, Auckland, New Zealand. (to be published).
- Lalonde, M. & Li, Y. (1997), Contour line extraction from color images of scanned maps, in 'ICIAP '97: Proceedings of the 9th International Conference on Image Analysis and Processing-Volume I', Springer-Verlag, London, UK, pp. 111–118.

- Losasso, F. & Hoppe, H. (2004), 'Geometry clipmaps: terrain rendering using nested regular grids', *ACM Transactions on Graphics* **23**, 769–776.
- Maia, M. A. G. M. & Átila L. F. Xavier (1996), 'A semiautomatic method for assigning elevation in contour maps', *IEEE Transactions on Knowledge and Data Engineering* **8**(4), 596–603.
- Marita, T. (n.d.), 'Border tracing algorithm'. URL: <http://users.utcluj.ro/~tmarita/IPL/L6/PI-L6e.pdf>.
- Peterson, S. (1998), 'Computing constrained delaunay triangulations in the plane'. URL: [http://www.geom.uiuc.edu/~samuelp/del\\_project.html](http://www.geom.uiuc.edu/~samuelp/del_project.html).
- Rivas, A. M. & de la Fraga, L. G. (2005), Terrain reconstruction from contour maps, in 'Proceedings of the International Workshop on Visualization and Animation of Landscape', Mexico City, pp. 167–175.
- Rognant, L., Planes, J. G., Memier, M. & Chassery, J. M. (2001), Contour lines and DEM: Generation and extraction, in 'Proceedings of First International Symposium on Digital Earth Moving (DEM 2001)', Lecture Notes in Computer Science, Springer-Verlag. URL: <http://www.springerlink.com/content/q962mm5bf2c43byd/fulltext.pdf>.
- Schiffman, H. R. (1996), *Sensation and Perception: An Integrated Approach*, 4<sup>th</sup> edn, John Wiley & Sons.
- TERRAINMAP.COM - Digital Elevation Modeling Journal (n.d.), 'Contour line extraction from a difficult topo map using BLACKART'. URL: <http://www.terrainmap.com/rm36.html>.
- U. S. Army Corps of Engineers and American Society of Civil Engineers (1996), *Photogrammetric Mapping*, ASCE Publications, New York.
- Wikipedia (n.d.), 'Contour map'. URL: [http://en.wikipedia.org/wiki/Contour\\_line](http://en.wikipedia.org/wiki/Contour_line).
- Xie, X. (2008), Automatic 3d terrain modelling from topographic map drawings, 780 project report, Dept. of Computer Science, University of Auckland, New Zealand.
- Xin, D., Zhou, X. & Zheng, H. (2006), 'Contour line extraction from paper-based topographic maps', *Journal of Information and Computing Science* **1**(5), 275–283.

# Measuring Visual Consistency in 3D Rendering Systems

Alfredo Nantes, Ross Brown and Frederic Maire

Queensland University of Technology  
Faculty of Science and Technology  
Brisbane, Queensland, Australia

a.nantes@qut.edu.au, r.brown@qut.edu.au, f.maire@qut.edu.au

## Abstract

One of the major challenges facing a present day game development company is the removal of bugs from such complex virtual environments. This work presents an approach for measuring the correctness of synthetic scenes generated by a rendering system of a 3D application, such as a computer game.

Our approach builds a database of labelled point clouds representing the spatiotemporal colour distribution for the objects present in a sequence of bug-free frames. This is done by converting the position that the pixels take over time into the 3D equivalent points with associated colours. Once the space of labelled points is built, each new image produced from the same game by any rendering system can be analysed by measuring its visual inconsistency in terms of distance from the database. Objects within the scene can be relocated (manually or by the application engine); yet the algorithm is able to perform the image analysis in terms of the 3D structure and colour distribution of samples on the surface of the object.

We applied our framework to the publicly available game *RacingGame* developed for Microsoft® Xna®. Preliminary results show how this approach can be used to detect a variety of visual artifacts generated by the rendering system in a professional quality game engine.

**Keywords:** Synthetic Image Analysis, Computer Vision, Computer Game Testing.

## 1 Introduction

One of the major challenges facing a present day game development company is the removal of bugs from such complex synthetic environments. Today, games provide remarkably realistic graphics with highly interactive scenarios held together by complex software that requires large development and testing teams. The more complex the software, the more crucial the effort of the companies in testing their product in order to ensure a high enough quality in terms of functionality, stability and robustness in general. Furthermore, a computer game is not only expected to work properly but it has to be, amongst other things, fun, challenging, realistic and well animated.

As observed by Macleod (2005), game play typically consists of a set of actions that move the game through a

number of successive states. Testing has the objective of ensuring that there is no combination of actions that brings the game to a state such that the experience meant to be offered gets corrupted. In recent years, researchers have been investigating the problem of defining and measuring the game play experience through cognitive models (Ermi & Mäyrä, 2005; C. A. Lindley & Sennersten, 2006; C. A. Lindley & Sennersten, 2007) or qualitative features (Kapoor, Burleson, & Picard, 2007; Roberts, Strong, & Isbell, 2007; Yannakakis & Hallam, 2007). These works have mainly attempted to model playability issues such as gameplay functionality, game usability and game mechanics.

In a previous work (Nantes, Brown, & Maire, 2008), we have classified these issues as the *Entertainment Inspection* component of game testing, distinguishing them from the complementary *Environment Inspection* issues. We defined the latter to be the degrees to which a game environment is perceived as being consistent in terms of sound, textures, meshes, lights and shadow effects that all contribute to the game play experience. An example of a visually inconsistent game environment is shown in Figure 1.

Focusing on the Environmental Integrity Inspection problem, we introduced an image processing approach to the automatic testing of such problems within 3D virtual environments and games (Nantes et al., 2008). The implementation of such an automated environment debugger can bring important benefits to the game industry. Indeed, it will bring costs savings by decreasing the number of play testers required during the Quality Assurance process. Moreover, it will increase the robustness of the product to release by allowing the coverage of a far larger set of test cases than currently performed by human players.

In this paper we present a complete and generalisable pixel transformation technique for the testing of 3D virtual environments that is feasibly able to be applied to any form of game engine that utilizes present rendering technology. It is generalisable, in being able to handle any scene using modern shader rendering, and enables the geometry and texturing in an environment to be tested automatically.

In Section 2 we review present work in the area of environment testing and geometry reconstruction. Section 3 presents our unique formulation to solve the scene integrity problem. The technique's theoretical framework is presented in Section 4. In Section 5 we present the algorithm we used for measuring the visual consistency between two frames. Section 6 concludes this paper with some remarks on preliminary results and future work.





Figure 1: Examples of visually inconsistent game environment. The pictures show some examples of mesh and texture corruption. Anomalies such as mesh corruption (present in all pictures) may occlude big areas of the screen, thus compromising the normal user-application interaction. Pictures (c) and (d) depict examples of texture corruption by which textures are wrongly mapped (the leaves of the palm trees in picture c are red; the car and the trees in picture d have transparent textures).

## 2 Related Work

Most work in automated game assessment is concentrated on the areas of game play via the use of artificial intelligence approaches to human entertainment modelling, for measuring qualitative factors such as challenge and curiosity (Yannakakis & Hallam, 2007) user interaction experiences (Yang, Marsh, & Shahabi, 2005), integrity and fairness in rudimentary computer games (Macleod, 2005), amongst others used to find faults in game mechanics (Chan, Denzinger, Gates, Loose, & Buchanan, 2004).

Although all these results could be used for assisting the design of the AI (Artificial Intelligence) component of a game, they do not address important environment integrity issues that surely affect the playability of the game environment. The small amount of Environmental Integrity Inspection research, has concentrated on algorithms for mesh geometry testing to find holes and slivers, (Cheng, Tamal, Edelsbrunner, Facello, & Teng, 1999) some techniques ending up in commercial systems such as, for example, the Half Life 2 Level Editor Hammer (Valve, 2009).

To our knowledge, no one else has attempted to use the image generated by a rendering system to reverse engineer the geometric contents of a scene, in order to perform geometry and texture integrity testing. In this work we propose a generalised image-space computer vision technique that addresses the automation of the Environment Integrity problem for a large class of geometry and texturing bugs.

Our new technique exploits the ability of the software rendering system to reverse the object transformations used to derive the final synthesized image. This inverse camera transform approach has similarities to computer vision techniques that exploit stereopsis and motion parallax from image data to generate 3D object geometry (Rusu, Blodow, Marton, & Beetz, 2008). These methods seek to generate the camera transform from noisy CCD imagery, to create a mesh representing the objects in the image. In these techniques, the control of accuracy is problematic, due to errors in the system caused by camera noise, sampling position estimation inaccuracies, and antialiasing caused by the limited sampling resolution available in video cameras.

In our technique the problems of correct geometry generation are largely removed, as we already have the transform that has been applied to the pixel in order to colour and position it in screen space. Since transforms

are already known, and we have control over the rendering environment, the original object geometry can be constructed from the series of test images presented, giving a superior approach to automated object geometry and texture checking, in the original normalized object coordinate system. We now proceed to describe this new inverse transformation technique in detail.

## 3 Problem Formulation

A picture can be defined by the way an observer perceives the distribution of colours on a specific support such as a piece of paper or a screen (Ning, 1997). The perception of a picture in terms of the meaning it has for the subject necessarily depends – among other things – on the observer experience (Giorgi, 2009).

According to this point of view, the correctness or consistency of a picture inherently depends on the observer perceptive capabilities and experience. Yet, a user-independent analysis can be done under the assumption that a number of visually consistent pictures can be generated – that is to say, images that are perceived as correct by some observer, e.g. the game designer.

By using a description similar to the one introduced by Fink et al. (2007), a game can be thought of as a two-function system. With the passage of time the game takes on a sequence  $s_0, s_1, s_2, \dots (s_i \in S)$  of states. Each state is obtained from the preceding one by the function

$$G_{\text{upd}} : S \times A \rightarrow S \quad (1)$$

where  $A$  is the input set of actions that can be performed by the player. The output of the game can be described by another function

$$G_{\text{out}} : S \rightarrow O \quad (2)$$

that produces the frame  $O$  from the current state  $S$ . Intuitively, the set  $O$  is correct if both  $S$  and  $G_{\text{out}}$  are correct.

In this work we assume we know that a collection of outputs in  $O$  was correct. We call such a collection the set of *validated frames* and we denote it by  $F_v$ . As we cannot say anything about the correctness of  $G_{\text{out}}$ , any output  $O$  not in the set of validates frames can be either correct or wrong. We call this collection the set of *test frames* that we denote by  $F_t$ . Note that the intersection of  $F_v$  and  $F_t$  may not be the empty set as  $S$  may contain repetitions.

With this work we aim to infer about the correctness of  $G_{\text{out}}$  via measuring the visual consistency of test frames

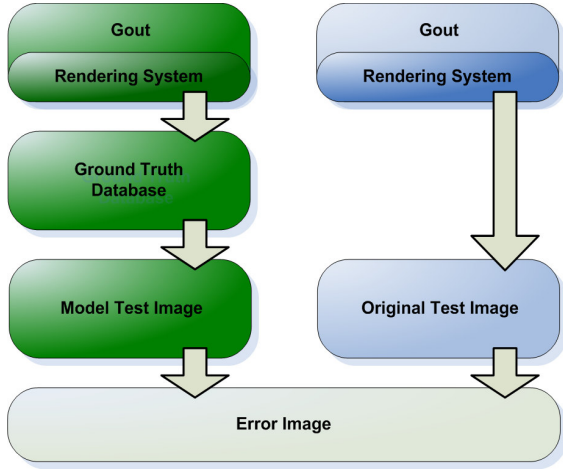


Figure 2: Framework for the detection of visual inconsistencies in test images. The left hand side of the diagram shows the process of building the ground truth database from validated images produced by some rendering system. Images are first converted to spatiotemporal colour distributions that form the ground truth database. Then models of the test images are generated for visual consistency measurements.

given a collection of validated frames and some information about  $G_{out}$  and its input  $S$ .

In this paper, unless otherwise specified, we will use  $G_{out}$  interchangeably to denote the output function and its hardware and/or software implementation.

#### 4 Theoretical Framework

If the test frames were produced from the same game states that produced the validated frames, the visual consistency of each frame in  $F_t$  would reduce to a mere pixel-by-pixel comparison with the related frame in  $F_v$ . Indeed, since the input states do not change and the output  $F_v$  is visually consistent by definition, any new output from  $G_{out}$  that differs from the related frame in  $F_v$  can be considered visually inconsistent.

Typically, the function  $G_{out}$  is partially implemented via software and partially via hardware. The hardware part corresponds to the GPU of the machine in which the game runs. Because a computer game is typically expected to run on a number of different GPUs, there will be a number of different implementations for some  $G_{out}$ . Therefore, it may well be that the same input state results in a different output frame for some  $G_{out}$ .

Instead of addressing the problem of measuring the correctness of  $G_{out}$  through a solution for reproducing a specific sequence of input states, we build a 3D model for each object that will be rendered in the collection of validated images  $F_v$  assuming that we know the set of geometric transformations that will be applied to the model by the rendering system. In addition to the RGB colour of the model in the frame, our validated models contain further information about the colour distribution by which they appear in  $F_v$ . Since the validated frames are correct, these models will represent the ground truth database for all objects appearing in a new frame. To put it another way, each model we generate is a spatiotemporal colour distribution that enables us – with

the due assumptions – to model even the appearance of animated or deformable objects, without knowing anything about the actual structure or dynamics of their components. Figure 2 depicts the overall process of error detection.

##### 4.1 Background

Any output frame is synthesized by  $G_{out}$  through a rendering process that takes objects in their original object or local space and turns them – via some geometric transformations – into the screen space to form the scene we perceive. Such a process is shown in Figure 3 wherein only the space transformation operations are depicted. Readers who are interested in a more comprehensive description of the rendering process can refer to Luna (2006).

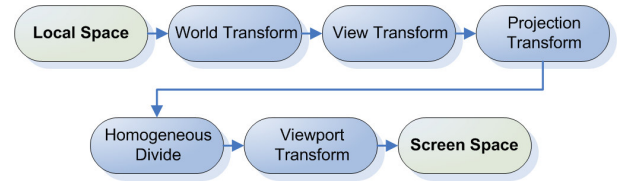


Figure 3: A GPU pipeline showing the process involved in transforming coordinates from local to screen space.

The *World* and *View* stages in the picture are the affine transformations that give the position in eye space. To represent affine transformation with matrices a homogeneous component  $w$  is introduced so that any vector  $(x, y, z)$  becomes  $(x, y, z, w)$  where  $w$  is typically set to 1. The *Projection* transformation then gives a position in a coordinate system bounded by the homogeneous unit cube: the *clip* space. Next, the position in normalized device coordinates is given by the *Homogeneous Divide*. This step brings the homogeneous component back to 1 after carrying out the projection matrix multiplication. Finally, the *Viewport* transformation stretches and translates the projected coordinates to fit a specific position of the screen. The final colour of a *fragment* – a portion of the final image of the size of a pixel – is decided by the lighting and shadowing techniques implemented by the designer as part of  $G_{out}$ .

The rendering process of Figure 3 is reversible in that, fragments of the screen can be brought back to their original position in object space. Let  $\mathbf{S}$  be the position of a fragment in screen space and  $\mathbf{M}$  the world-view-projection matrix that transforms the object to which the fragment belongs from original position to screen space position. The following relations hold:

$$\mathbf{D} = \mathbf{S} \times \mathbf{M}_v^{-1} \times \mathbf{M}^{-1} \quad (3)$$

$$\mathbf{R} = \frac{\mathbf{D}}{\mathbf{w}} \quad (4)$$

where  $\mathbf{M}_v$  is the *viewport* matrix of the frame;  $\mathbf{R}$  is the original position of the fragment equivalent to  $\mathbf{S}$  given  $\mathbf{M}$  and  $\mathbf{M}_v$  and  $\mathbf{w}$  is the homogeneous coordinate component of  $\mathbf{D}$ . We denote a fragment in its original

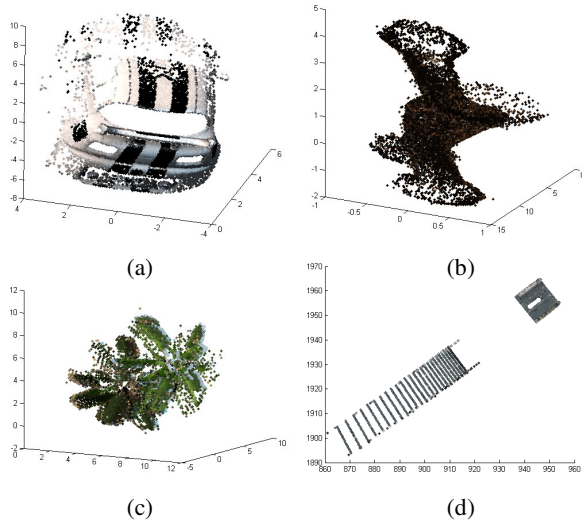


Figure 4: Examples of vectors  $(\mathbf{R}, \mathbf{C})$  for four different objects observed in a collection of frames, namely a car (a), the trunk of a palm tree (b), the leaves of a palm tree (c) and a segment of the track (d).

position in object space by the vector  $(\mathbf{R}, \mathbf{C})$  where  $\mathbf{C}$  is the final colour that the fragment takes at the end of the rendering process. Note that the perspective-projection matrix is, in general, singular due to the lost of the  $z$  dimension; individual points in eye space that lie along the same line of projection will project to a single point. However, since rendering systems map the coordinate values within the view volume to a (unit) cube, the information about  $z$  is kept and the matrix becomes invertible (Van Verth & Bishop, 2004). Finally, note from Figure 3 that the *rescaling* operation performed by Equation 4 should take place after inverting the Projection transform and before multiplying by the inverse of the World-View matrix. However, because no affine transformation (World and View) can alter the homogeneous component of a coordinate vector, this operation can be performed at the end of the reversing process. The advantage of postponing the rescaling operation is to compute the inverse of two matrices (the Viewport and the combined World-View-Projection) instead of three (Viewport, Projection and the combined World-View).

#### 4.2 Visual Consistency Error in Object Space

By applying Equation 3 and 4 to all fragments of an object in a frame we end up with a vector of 3D coloured points that model the spatial and colour distribution of those parts of the object visible in that frame. The colours



Figure 5: Example of *object map* (left) and *matrix map* (right) of a frame. Colours identify different objects in the object map and different world-view-projection matrices in the matrix map. The depth buffer is encoded by the  $\alpha$  channel of the object map.

are taken directly from the final image, the *frame buffer*. As long as  $\mathbf{S}$ ,  $M_v$  and  $M$  are known  $(\mathbf{R}, \mathbf{C})$  can be built from an arbitrary number of frames. Figure 4 depicts four examples of the vector  $(\mathbf{R}, \mathbf{C})$  for different objects that have been built from a number of frames. Such objects form the ground truth database of Figure 2 by which the test images will be assessed.

#### Definition 1: Visual Consistency of an Object in Object Space

Given a set  $A$  of fragments of an object in object space relative to a single frame and the set  $B$  of fragments of the same object in object space relative to some collection of frames;  $A$  is  $\epsilon$  consistent with  $B$  in object space if

$$d_o(A, B) \leq \epsilon \quad (5)$$

We call  $d_o(A, B)$  the *visual consistency error* of  $A$  relative to  $B$  in object space.

Henceforth,  $\epsilon$  is assumed to be a small constant.

#### Definition 2: Visual Consistency of a frame in Object space

A frame  $F_i$  is  $\epsilon$  consistent with a collection of frames  $F_v$  in object space if the set  $A$  is  $\epsilon$  consistent in object space with the set  $B$  for all objects in  $F_i$ .

#### 4.3 Building the vector $(\mathbf{R}, \mathbf{C})$ through the game engine

From Definition 2 it follows that in order to decide whether a test frame is correct or not we need to compute the vectors  $(\mathbf{R}, \mathbf{C})$  for all objects appearing in that frame and in some given collection of frames. Such process, entails  $\mathbf{S}$  and  $M$  to be known for each object, for each frame. Without loss of generality, we can assume that  $M_v$  is constant for all frames generated by  $G_{out}$ .

During the rendering process a graphics application (such as a computer game) passes the objects that need to be rendered to the GPU as well as – among other things – their geometric transformations. The screen space coordinates of the fragments are then computed by the GPU. Therefore, to determine both  $\mathbf{S}$  and  $M$  for each object we need the GPU to label with  $f(O)$  those fragments of the current frame belonging to the object  $O$  and with  $g(M)$  those fragments generated from the transformation  $M$ . The functions  $f$  and  $g$  should produce a unique colour identifier given an object or a matrix as input. By doing so we effectively generate two images that allow us to map the colours of  $\mathbf{S}$  and  $M$  to the related object  $O$  and matrix  $M$ . We call the image generated through  $f$  the *object map* and the image generated through  $g$  the *matrix map*.

To make this process as little game software dependent as possible we created a class *LabelMapShader* that inherits from the same class used to initialize any other shader effect in the game. As with the other render classes of the game, our class is instantiated once and used whenever an object needs to be rendered. The label map class functionality is turned on and off by



Listing 1. Pseudocode of our *LabelMapShader* class.**Attributes:**

*OM, MM*: TEXTURE  
*objColour, mtxColour*: EFFECT\_PARAMETER  
*objColTab*: TABLE<String, Vector>  
*mtxColTab*: TABLE<Matrix, Vector>

**Methods:**

*updateMaps(O, M)*  
  set *objColour* to  $f(O)$   
  **if** (*O.name* not in *objColTab*)  
    add to <*O.name*, *objColour*> to *objColTab*  
  **end**  
  set *mtxColour* to  $g(M)$   
  **if** (*M* not in *mtxColTab*)  
    add <*M*, *mtxColour*> to *mtxColTab*  
  **end**  
  set render targets to *OM* and *MM*  
  commit changes to the GPU

*generateMaps(O)*  
  set *LabelMapping* to be the current shader technique  
  render *O*

debug flags in the game source code. The pseudocode of such a class is shown in Listing 1. Object and matrix maps – identified by the attributes *OM* and *MM* respectively – are implicitly updated by the function *generateMaps()*. When an object needs to be rendered the game first calls the function *updateMaps()* which adds the object name and matrix along with the associated colours to the tables *objColTab* and *mtxColTab*. In this way, colours are linked to the related objects in the object map and to the related matrices in the matrix map. Then the *updateMaps()* function sets the colours to be used by the shader for rendering the object *O* to the two textures *OM* and *MM*. The function *generateMaps()* – which is called by the game right after the function *updateMaps()* – sets the shader technique to

Listing 2. Shader Code that generates the object and matrix maps for the current frame.

```
VSOutput OMMMap_VS(VSInput In)
{
  VSOutput outVS = (VSOutput)0;
  outVS.Pos = mul(float4(In.Pos.xyz, 1.0f),
                  worldViewProj);
  outVS.Depth = float2(outVS.Pos.z,
                      outVS.Pos.w);
  return outVS;
}

PSOutput OMMMap_PS(VSOutput In)
{
  PSOutput outPS = (PSOutput)0;
  outPS.ObjCol.xyz = objColour.xyz;
  outPS.ObjCol.w = (In.Depth.x)/(In.Depth.y);
  outPS.MtxCol = mtxColour;
  return outPS;
}
```

use whose code is shown in Listing 2. Then it sends the object *O* to the GPU to have it rendered on the texture *OM* with the colour *objColour* and to the texture *MM* with the colour *mtxColour*.

The function  $f$  we used simply returns a four-byte identifier set to the current value of an internal counter. The counter is incremented only if the name of the next object to render is not already in the table *objColTab*. Bytes are then treated as  $(R, G, B, \alpha)$  colour components. Of this vector, only the  $(R, G, B)$  components are used for colouring the pixels. The  $\alpha$  channel is replaced by the value of the depth buffer which is computed by the same shader code used to generate the maps. This is shown in Listing 2. As it can be noted, the pixel shader renders to two targets, namely *OM* and *MM* corresponding to the object map and matrix map texture respectively. The variable *objColour* containing the  $(R, G, B)$  components of the colour to be rendered is stored in the first three channels of the target *OM*. The forth channel of *OM* is set to the normalized  $z$  value of the screen space position computed by the vertex shader, namely the depth of the pixel. Therefore, the object map texture completely defines *S* through the table *objColTab*. Such a texture represents the final image segmented at the object level. The colour of the segments is specified in *objColTab*.

For the matrix map, fragments of the object that have undergone the transformation *M* take the colour generated through the function  $g$  and stored in the table *mtxColTab*. The colour returned by  $g$  is a four-byte array set to the current value of an internal counter. This counter is incremented only if the matrix of the next object to render is not already in the table *mtxColTab*. This time all  $(R, G, B, \alpha)$  components of the array are used for colouring the pixels. Figure 5 depicts an example of such maps related to a frame. Note that, with this implementation, the object map can map up to  $2^{3d}$  objects per frame where  $d$  is the colour depth used. For a 32-bit *truecolor* image (8 bits per channel) the maximum number of objects that can be stored for a single frame is  $\approx 1.6 \cdot 10^7$ . Likewise, the maximum number of transformation matrices that can be mapped by the matrix map is  $2^{4d}$  that is  $\approx 4.3 \cdot 10^9$  matrices for a *truecolor* image.

#### 4.4 Visual Consistency Error in Screen Space

The process described in the previous Section can be used for computing the vector  $(\mathbf{R}, \mathbf{C})$  of an object in object space, given a frame or a set of validated frames in which the object appears. This requires the normal rendering process of  $G_{\text{out}}$  to be modified. Indeed, two additional steps need to be performed for each object to be rendered namely, the building of the object and matrix maps. If the rendering pipeline cannot be modified during testing the visual consistency can still be measured in screen space. To that end, the vector  $(\mathbf{R}, \mathbf{C})$  needs to be converted to the equivalent vector  $(\mathbf{S}, \mathbf{C})$  where *S* is the screen space position equivalent to *R*. This can be done by using the same geometric transformations used by the

game to render the test frame. Since  $(\mathbf{R}, \mathbf{C})$  models the objects rendered by the game, this process will create a model of the test frame. For this reason, we shall call the vector  $(\mathbf{S}, \mathbf{C})$  the *model test frame*.

This transformation process, shown in Listing 3, consists of an emulation of the rendering process in the graphics pipeline and assumes that some information about the test image is available. Apart from the vector  $(\mathbf{R}, \mathbf{C})$ , also the transformation matrix  $M$ , the viewport matrix  $M_v$  and the depth buffer  $Z$  of the test frame need to be known *a priori*. Note that  $M$ ,  $M_v$  and  $Z$  can be extracted from the input and output of the rendering pipeline without modifying the rendering process itself.

After computing the normalized device coordinates through the function *normalize()*, points outside the normalized volume of space (*frustum*) are clipped through the binary mask generated by the function *xyzClipMask()*. To be sure, also the colour information about the points needs to be updated. This is done by masking the colour vector with the same *cMask* used for clipping the device coordinates.

Once the device coordinates have been computed the GPU performs the *backface culling* (Luna, 2006). However, as our object model  $(\mathbf{R}, \mathbf{C})$  is made of points and not polygons, there are no faces to remove but sets of points that model them. Assuming that we have the depth buffer  $Z$  of the test frame we can use it as *z-test* function. That is, we remove from the vector *scrCoords* those points whose *z* value does not match with the value of the depth buffer at screen position  $(scrCoords.x, scrCoords.y)$ . The variable *zMask* is then a binary vector whose elements are 1 if the relation

$$Z(scrCoords.x, scrCoords.y) = scrCoords.z \quad (6)$$

holds; 0 otherwise.

When computing  $\mathbf{R}$  in Equation 4 the approximated version of  $\mathbf{S}$  is used as the screen space coordinates are read from the image and the depth buffer, both of finite resolution. This produces the aliasing effect of scattered points visible in Figure 4. Hence, because of the discretization error introduced when building  $(\mathbf{R}, \mathbf{C})$ , Equation 6 needs to be modified so as not to clip too many valid points from the *scrCoords* vector. That is, we need to allow for some tolerance  $\tau$  when performing the *z*



Figure 6: Relationship between a fragment in screen space and the related region of the vector  $(\mathbf{R}, \mathbf{C})$ . The cube in object space is the equivalent region of the fragment at screen position  $(x, y)$ . The number of points  $p$  in the cube corresponds to the number of points in the model test frame at  $(x, y)$ .

Listing 3. Pseudocode for converting the vector  $(\mathbf{R}, \mathbf{C})$  to the vector  $(\mathbf{S}, \mathbf{C})$ .

**function** *transformRC*(( $\mathbf{R}, \mathbf{C}$ ),  $M$ ,  $M_v$ ,  $Z$ ,  $\tau$ ) **returns** ( $\mathbf{S}, \mathbf{C}$ )

```

devCoords  = normalize( $\mathbf{R} * M$ )
cMask      = xyzClipMask(devCoords)
devCoords  = devCoords(cMask)
colorSet   =  $\mathbf{C}(cMask)$ 
scrCoords  = devCoords *  $M_v$ 
zMask      = zTest(scrCoords,  $Z$ ,  $\tau$ )
scrCoords  = scrCoords(zMask)
colorSet   = colorSet(zMask)

```

**return** (*scrCoords*, *colorSet*)

test. Because of the non-linearity of the depth buffer this tolerance will be a function *zTest()* of  $\tau$  and the depth of each pixel in the image which will be read from the depth buffer  $Z$ . The result of this last operation is the screen space vector  $(\mathbf{S}, \mathbf{C})$  equivalent to  $(\mathbf{R}, \mathbf{C})$ .

It is important to note that the model test frame may not be the exact copy of the test frame. More precisely, a fragment of the test frame at screen position  $(x, y)$  coincides with a number  $p$  of fragments of the model at the same position. The number  $p$  depends on the density of the vector  $(\mathbf{R}, \mathbf{C})$  in the object space region equivalent to the pixel  $(x, y)$ . Such a density, in turn, depends on the distances  $z$  in screen space at which the same region of  $(\mathbf{R}, \mathbf{C})$  has been observed in the validated frames; the bigger such a distance, the sparser the region.

The relation between  $p$  and the density of  $(\mathbf{R}, \mathbf{C})$  is illustrated in **Error! Reference source not found.** If  $p$  is zero, the related pixel of the model test frame will not contain fragments from  $(\mathbf{R}, \mathbf{C})$ . Empty regions of the model test frame will be further discussed in Section 5.

Finally, it should be noted that the colour of the  $p$  fragments at  $(x, y)$  may not be the same as the colour of the pixel at the same position. This is due to the different light conditions under which the same region of  $(\mathbf{R}, \mathbf{C})$  has been observed in the validated images.

Once we have the vector  $(\mathbf{S}, \mathbf{C})$  we can measure the visual consistency in screen space.

### Definition 3: Visual Consistency of an Object in Screen Space

Given a set  $A$  of fragments of an object in screen space relative to a single frame and the set  $B$  of fragments of the same object in screen space relative to some collection of frames;  $A$  is  $\epsilon$  consistent with  $B$  in screen space if

$$d_s(A, B) \leq \epsilon \quad (7)$$

We call  $d_s(A, B)$  the *visual consistency error* of  $A$  relative to  $B$  in screen space.

Listing 4. Pseudocode for measuring the visual consistency of a test frame.

---

```

function measureVC(A, B) returns two images

    local variables: qThres, quadTree threshold
                     dThres, node density threshold
                     r, patch radius

    frBuffer = frame(A)
    setQ = qTreePartition(frBuffer, qThres)
    for each node n in setQ
        colVectN = colour(A, n)
        mColourN = mean(colVectN)
        if density(B, n) < dThres
            nCentre = centroid(n)
            build a patch P of radius r centered at nCentre
            setP = points of B.S in P
            if setP is empty
                vConst(n) =  $\infty$ 
                spDist(n) =  $\infty$ 
                continue
            end
            setO = kNN of nCentre among the points of setP
            distO = mean of distances from nCentre to setO
        else
            setO = points of B.S in n
            distO = 0
        end
        colVect = colour(B, setO)
        setC = kNN of mColourN among the colours of colVect
        mu = mean(setC)
        sigma = covariance(setC)
        vConst(n) = mgd(nColour, mu, sigma)
        spDist(n) = distO
    end
    return vConst, spDist

```

---

#### Definition 4: Visual Consistency of a Frame in Screen Space

A frame  $F_i$  is  $\mathcal{E}$  consistent with a collection of frames  $F_v$  in screen space if the set  $A$  is  $\mathcal{E}$  consistent in screen space with the set  $B$  for all objects in  $F_i$ .

### 5 Visual Consistency Measurements

This section presents an algorithm for measuring the visual consistency error in screen space. Before reviewing the algorithm, however, it is important to clarify the concept of density of the model test image and explain why it is factored into the error measurement process.

Consider the model test frame in Figure 7b. By comparing it with the original test frame, it can be noticed that the model has sparse or even empty regions. As argued in Section **Error! Reference source not found.**, sparse or empty regions in  $(S, C)$  may be due to equivalent sparse or empty regions in the vector  $(R, C)$ . However, an empty region in the model can also be caused by an object that does not have a related  $(R, C)$

because it is not present in the collection of validated frames. Unfortunately, if an object is not present in such a collection we cannot claim that the related empty region is an anomaly as we are not assuming that the validated frames contain all possible objects of the game. Nor should we claim that the region is correct as the information we need for measuring the visual consistency is missing. We can then say that empty regions in the model test image are areas of high uncertainty with respect to the visual consistency error that can be measured on them. Conversely, a dense region of the model is a region that has been observed from the same or a closer distance in the collection of validated frames and/or from different camera angles. Hence, the measure of its visual consistency is expected to be accurate. Dense regions in the model test frame are then areas of high certainty. The sparseness of a region in the model is therefore a measure of confidence with which we should accept the visual consistency error computed on the same regions.

In this work we used the Multivariate Gaussian Distribution (MGD) to measure the visual consistency error in screen space. However, instead of computing the distance for each objects in the test frame, we computed the distance for rectangular supports of a quadtree subdivision. Listing 4 shows the pseudocode of our implementation.

After extracting the test frame from  $A$  through the function *frame*() a quadtree partitioning is applied to it.

A quadtree node (quad) is no longer sub-divided if the difference between the maximum and the minimum values of each colour component in the node is smaller than its respective threshold *qTrhes*. In our experiments such a threshold was set to 0.5 for all colour components. We found this being a good trade-off between speed and accuracy. The output of the partitioning is the set *setQ* of nodes that make up the quadtree. Original and model test frame share the same partitioning.

The node density – computed by the function *density*() – is a measure of the sparseness of the model  $B$  for the current node. However, because nodes have been partitioned on the basis of their colour homogeneity and not their sparseness, small nodes in sparse (non-empty) regions may have zero density, which would be interpreted as maximum uncertainty. To allow for this, we build a patch of radius  $r$  that surrounds the low density node. If the patch still does not contain  $(S, C)$  points the node is considered empty and it will be assigned the maximum distance and uncertainty. Otherwise, to find the most appropriate surrounding  $(S, C)$  points by which to measure the consistency error we use the k-Nearest Neighbours (kNN) algorithm. Specifically, we compute the  $k$  nearest model points in the neighbourhood of the node centre where  $k$  is the minimum value between 10 and the set of model points in the patch. By storing the mean radius of such a set, the variable *distO* represents the confidence in making any inference about the node. If the node is dense, there are enough  $(S, C)$  points to compute the visual consistency error and to expect it to be accurate (*distO* = 0).

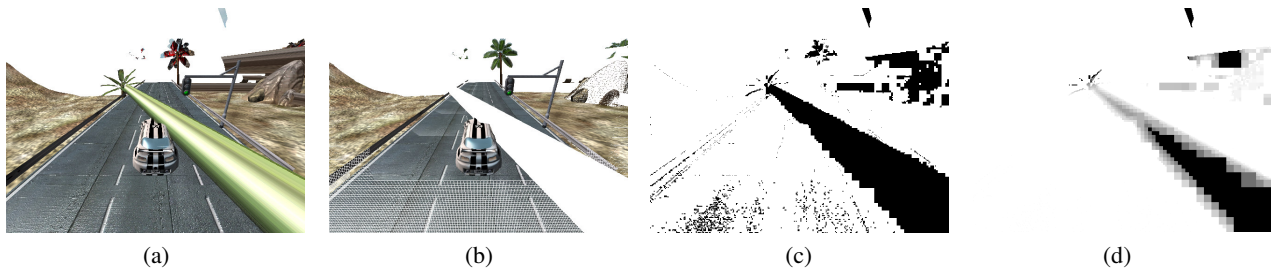


Figure 7: Example of original test frame (a), model test frame (b), visual consistency map (c) and confidence map (d). Sparse regions in the model test frame correspond to equivalent regions of low density in the vector ( $\mathbf{R}, \mathbf{C}$ ). Black regions in the Visual Consistency Map correspond to high inconsistencies with respect to the vector ( $\mathbf{S}, \mathbf{C}$ ). Dark regions in the Confidence Map correspond to areas of low confidence about the consistency measure.

Finally, the error is computed with the  $k$  colours of the model test frame closest to the mean colour of the node. The parameter  $k$  is set to be the minimum value between 10 and the number of colours in *ColVect*. From the set of colours extracted, the mean and variance are computed. The consistency error is returned in terms of likelihood – through the multivariate probability density function – of the mean colour of the node given the spatially close fragments observed in the collection of validated frames. Low probabilities correspond to high inconsistencies.

The algorithm generates two images namely, *vConst* encoding the visual consistency error for all nodes of the partitioned frame; and *spDist* representing the confidence about the consistency measured for each node. We call the image *vConst* the *Visual Consistency Map* and the image *spDist* the *Confidence Map*.

Figures 7c and 7d show the output of the algorithm for the test image in Figure 7a. In order to make the *vConst* image visually interesting we only displayed those nodes whose consistency error was below a certain threshold ( $1 \cdot 10^{-6}$  in this case). Areas of high uncertainty (dark regions) in the Confidence Map are located in correspondence of empty or sparse regions of the model test frame. The original test frame in Figure 7 is an example of a buggy test image affected by texture corruption (the texture of the palm tree leaves in the background is wrong) and polygon corruption (polygons from the small palm tree and the rocks in the background are wrongly projected). As can be observed from the output of our algorithm such artifacts are correctly detected.

## 6 Conclusions

This work introduces a general approach for measuring visual inconsistencies in synthetic images generated by a graphics application, such as a computer game. The approach is independent from the 3D application, as it mainly relies on the standard input and output data of the rendering system, and does not require a large amount of source code modification to be implemented within a typical game engine.

Although the accuracy of our implementation is yet to be measured, preliminary results show that the framework is able to cope with geometry and texturing anomalies such as polygon and texture corruption without requiring any knowledge about the geometric primitives and the textures used by the application. It is postulated that the algorithm presented in Listing 4 will perform well for those environments where the colour and geometry of a

single object does not (sensibly) depend on the rest of the scene. Colour changes due to *global effects* such as shadows and light reflections and refractions are unlikely to be effectively captured or measured by our algorithm. If the colour or the geometry of fragments sensibly varies over time a more complex mechanism is required. Such a mechanism should be able to elicit possible causal relationships out of specific visual events in the set of validated images and allow for those causes to make inference about similar events in new images.

Our framework has been applied to the publicly available game *RacingGame* a close to professional quality game engine that has been developed as *starter kit* for Microsoft® Xna®<sup>1</sup>. Such a game turned out to be a good test bench for our framework for the following reasons:

- due to its prototypal nature, all visual inconsistencies that we have observed were generated by the game and never induced by us;
- the world position of some objects in the environment (like buildings, banners and trees) was automatically changed by the game at each play session. This allowed us to test our framework for volatile virtual environments;
- render effects are such that the colour and geometry of the objects does not sensibly depend on the rest of the scene.

As far as concerns the game we have tested, the consistent appearance of the objects was correctly discriminated from other visual inconsistencies.

Our experiments show that the Multivariate Gaussian Distribution gives a good measure of the visual inconsistency error for high density regions of the model test frame. However, it can be noticed that sparse regions in the model are likely to be classified as bugs regardless the actual consistency of the equivalent areas in the original test frame.

In our future work we will investigate the use of other functions for measuring the inconsistency error such as the *Kullback-Leibler* divergence and the *Hausdorff* distance. Also, we mean to effectively combine the Visual Consistency Map with the Confidence Map in order to improve the robustness of the solution. To this end, we will build a Bayesian detector from these two maps. After segmenting the visual consistency map into inconsistent regions, we will weigh the inconsistent

<sup>1</sup> <http://creators.xna.com/en-US/education/starterkits/>



regions with respect to the confidence map. Those regions whose total weights are above a threshold will be flagged as likely to be bugs.

Finally, we mean to provide a consistent way of measuring the accuracy of our algorithm for a proper validation via testing with a larger set of varying scenes, first evaluated for errors by subjective viewers.

## 7 References

- Chan, B., Denzinger, J., Gates, D., Loose, K., & Buchanan, J. (2004). *Evolutionary behavior testing of commercial computer games*. Paper presented at the 2004 IEEE Congress on Evolutionary Computation.
- Cheng, S., Tamal, K. D., Edelsbrunner, H., Facello, M. A., & Teng, S. (1999). Sliver Exudation. *Journal of the ACM*, 47(5), 883 - 904.
- Ermi, L., & Mäyrä, F. (2005). *Fundamental Components of the Gameplay Experience: Analysing Immersion*. Paper presented at the Digital Games Research Association (DiGRA-05), Perth, Australia.
- Fink, A., Denzinger, J., & Aycok, J. (2007). *Extracting NPC behavior from computer games using computer vision and machine learning techniques*. Paper presented at the IEEE Symposium on Computational Intelligence and Games, Hawaii.
- Giorgi, A. (2009). The Phenomenological Mind: An Introduction to Philosophy of Mind and Cognitive Science. *Journal of Phenomenological Psychology*, 40(1), 107-108.
- Kapoor, A., Burleson, W., & Picard, R. W. (2007). Automatic prediction of frustration. *International journal of human-computer studies*, 65(8), 724-736.
- Lindley, C. A., & Sennersten, C. C. (2006). *A Cognitive Framework for the Analysis of Game Play: Tasks, Schemas and Attention Theory*. Paper presented at the 28th Annual Conference of the Cognitive Science Society.
- Lindley, C. A., & Sennersten, C. C. (2007). Game Play Schemas: From Player Analysis to Adaptive Game Mechanics. *International Journal of Computer Games Technology*, 2008(2).
- Luna, F. (2006). *Introduction to 3D Game Programming with Direct X 9.0c: A Shader Approach (Wordware Game and Graphics Library)*. Plano, Texas: Wordware Publishing Inc.
- Macleod, A. (2005). *Game design through self-play experiments*. Paper presented at the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology.
- Nantes, A., Brown, R., & Maire, F. (2008, 22-24 October). *A Framework for the Semi-Automatic Testing of Video Games*. Paper presented at the Artificial Intelligence and Interactive Digital Entertainment (AIIDE-08), Stanford University, Palo Alto, California.
- Ning, L. (1997). *Fractal Imaging*. San Francisco, California: Morgan Kaufmann Publishers Inc.
- Roberts, D., Strong, C., & Isbell, C. (2007). *Estimating Player Satisfaction Through the Author's Eyes*. Paper presented at the Workshop on Optimizing Player Satisfaction Artificial Intelligence and Interactive Digital Entertainment (AIIDE-07), Stanford University, Palo Alto, California.
- Rusu, R. B., Blodow, N., Marton, Z. C., & Beetz, M. (2008). *Aligning Point Cloud Views using Persistent Feature Histograms*. Paper presented at the 21st IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Nice, France.
- Valve. (2009). Valve Developer Community. Retrieved June 2nd, 2009, from [http://developer.valvesoftware.com/wiki/Main\\_Page](http://developer.valvesoftware.com/wiki/Main_Page)
- Van Verth, J., & Bishop, L. (2004). *Essential Mathematics for Games and Interactive Applications: A Programmer's Guide*. San Francisco, California: Morgan Kaufmann Publishers Inc.
- Yang, K., Marsh, T., & Shahabi, C. (2005). *Continuous archival and analysis of user data in virtual and immersive game environments*. Paper presented at the ACM workshop on Continuous archival and retrieval of personal experiences.
- Yannakakis, G. N., & Hallam, J. (2007). Towards Optimizing Entertainment in Computer Games. *Applied Artificial Intelligence*, 21(10), 933-971.



# Fluid Dynamic Visualisations of Cuttings-Bleeding for Virtual Reality Heart Beating Surgery Simulation

Sugeng Rianto<sup>1</sup>, Ling Li<sup>2</sup>

<sup>1</sup>Department of Physics, Brawijaya University Malang – Indonesia 65145

<sup>2</sup>Department of Computing, Curtin University of Technology  
Perth - Australia 6102

priantos@brawijaya.ac.id

## Abstract

Visualisations and computations of a real time fluid dynamic rendering have always been a fascinating research topic in the field of computer graphics. However most existing virtual realities surgery simulators tend to avoid the fluid dynamic model involvement in their system. This is due to the fact that real time fluid dynamic visualisation is computationally expensive. The calculation may slow down the rendering time and reduce force feedback interactions during simulations. With the availability of high performance graphic hardware, the improvement of visual quality and computational accelerations become easier to achieve. In this paper, we propose fast and efficient fluid dynamic visualisations for a heart surgery simulation. The algorithm utilizes the GPUs capable streaming computations to generate physically-based computational fluid dynamic for bleeding in real time. Our approach is based on the Navier-Stokes Equations that is implemented on two-dimensional data structure which stores height fields, blood quantity, and dissolving blood velocities. We use the cubic interpolated propagation as a fluid solver of blood movement. In this paper, the blood flowing on the surface of a beating heart when the surgical knife cut the heart muscle surface in a certain thickness. The blood will move from the sources and follow the height map of the heart surface. The comparison of the frame rates of surgery simulation with and without fluid inclusion is presented and proves that our approaches are effective enough for evolving fluid dynamic visualisation during surgical simulations.

**Keywords:** GPU, heart, surgery simulation

## 1 Introduction

Although the introduction of minimally invasive techniques and new stabilizing devices has undergone resurgence of the beating heart of the coronary artery bypass, the development of a method for training surgeons to perform accurate anastomoses despite cardiac movement and to develop the skills are still needed for keeping a reliable result (Stanbridge, David O'Regan et al.

1999). Consequently, simple methods to train surgeons in performing these operations are urgently needed. While the application of robot-assisted coronary artery surgery has been introduced, more advance technology may be involved to reach optimal results.

In the last few years, the developments of computer graphic technology and hardware have evolved into an effective tool not just in generating more realistic visualization but also in improving the speed of computation. A virtual reality (VR) simulation now becomes possible to be established in real time with force feedback interactions. This fact bring VR to be widely used as a training tool in many applications (Stone 2000). One of the potential applications is in medical industries, particularly for surgery simulations. This techniques can provide a realistic, safe, and controllable environment for novice doctors to practice surgical operations, allowing them to make mistakes without serious consequences (Anonymous 2001; Higgins and Koucky 2002).

Fluid dynamic modelling can enhance the realism in surgical simulation. Without fluid, the surgical simulator is dry and clean. However, most surgery procedure involves fluid. Particularly in heart surgery the fluid interaction during simulation is important. Although fluid dynamic inclusion may have not much effect to force feedback during the simulation, the existing fluid cannot be avoided (Bhasin, Liu et al. 2005) during surgical operations. Besides, it is providing a realistic visual cue, since the existing fluid can obscure surgeon's view and make operation difficult. The fluid model can be useful in trainings such as sucking, draining, and flooding, or tackling unpredictable blood splashing from the heart.

The fast improvements and its programmability of graphics hardware (Owens, Luebke et al. 2005) allow us to fulfil computationally demanding tasks in many purposes. This has led us to develop a new mechanism in designing surgical simulator. The simulator includes computation of fluid movement based on General-purpose computation on graphics processors (GPGPU). Our approach provides the balancing between computation and real-time visualisations for designing user interactivity during surgical simulation designs.

This paper proposes force and torque models for generating feedback on both the fluid and heart muscle movement into a haptic device during heart beating surgery simulations. The users not only see the fluid, but also can feel and interact with them (blood and water). We introduce fast solutions of Navier-Stokes Equations (NSEs) using a modified cubic interpolated propagation

scheme. The framework is used to provide real-time computations with realistic 3D visualizations.

## 2 Related Work

In the last few years, many approaches have been developed to compute fluid dynamic models. Some of them also have done research in the area of fluid dynamic simulation and virtual reality. This section discusses some previous work and will divide discussion into four sections: fluid modelling, haptic renderings, fluid dynamic solver, and GPU programming.

### 2.1 Fluid modelling for computer graphic

Recently, the modelling of fluid dynamic has received much attention in the computer graphics society. Although, there are many good works related to this field, this paper only introduce some recent closely related works.

A fluid modelling in computer graphic with interactive animation was pioneered by (Stam and Fiume 1993). This study simulated a fluid turbulence based on advection-diffusion model of particle-based gaseous phenomena. They used clustering algorithm to get efficient fluid animations and renderings. This work had been extended by (Foster and Metaxas 1996) with introduction of a more complex fluid behaviour. However, those studies suffer from blowing when a bigger time step is applied. A semi-Lagrangian method with unconditionally stable fluid simulation, then, was introduced by (Stam 1999) to overcome that problem. Some other researches to augment fluid modelling such as: particle-based fluid animation (Kruger, Kipfer et al. 2005; Muller, Solenthaler et al. 2005) and turbulent water over natural terrain had been continuously developed for various applications. All of these studied mainly discuss on the ways of finding efficient NSEs solution of motion for liquid with realistic looking behaviours for practical animations. Other methods as proposed in (Matthias and Ller 2009) for tracking fluids over surface, in (Adrien, Andrew et al. 2006) for real time interaction between fluid and objects, and in (Benjamin and Erik 2009) for real-time fluid simulation, look superior than the proposed method; however, these methods only interact with mouse or digitally generated object. Their robustness, real-time interaction, and its computation efficiency, have not been tested for multisensory hardware interaction such as haptic.

In the last ten years the introduction of graphic processor unit (GPU) has also embossed fluid modelling in several ways. GPU not just provide better visualizations, but also support efficient computations with its parallelism and programmability. The application of GPU for flow simulation was introduced by (Csebfalvi and Szirmay-Kalos 2003). This study not only focuses on visual fidelity as done in (Stam and Fiume 1993; Foster and Metaxas 1996; Stam 1999; Muller, Solenthaler et al. 2005; Losasso, Irving et al. 2006), but also attempts to accelerate the flow simulation in real time speed, while maintaining physical accuracy. The parallel algorithm to approach shallow water equation on GPU as extension of (Kass and Miller 1990) work was implemented by (Noe and Trier 2004). Moreover, the

GPU applications for interactive fluid motion in terrain rendering with erosions (Anh, Sourin et al. 2007; Mei, Decaudin et al. 2007) also become a current research interest in computer graphics and animation. While these studies show potential outcome in increasing the visual quality and the user interactivity, none of them involved force feedback interactions in their fluid visualizations.

### 2.2 Haptic rendering for fluid interaction

The exploration of force feedback design and computations for haptic instruments has been done for several years; however, there are only a few that concern with force feedback design for fluid media. Haptic rendering method was introduced by (Avila and Sobierajski 1996). This study, then, inspired many applications such as: surgical cutting or trimming (Thomas V and Cohen 1999), sculpting (Kim and Park 2004), and volume visualization (Lundin, Gudmundsson et al. 2005). Whereas a force feedback integration with interactive fluid model has been introduced in (Baxter and Lin 2004). This method enables force and torque generations in virtual painting applications. There is no other study exploring this field except in (Lundin, Sillen et al. 2005) for presenting fluid dynamic data and in (Bhasin, Liu et al. 2005) for simulating droplet fluid. Both of them more concentrate on the fluid model visualization rather than its interaction to haptic device.

### 2.3 The development of fluid solver

There many floating numerical fluid dynamic solvers have been developed; however, the discussion in this section only focuses on the numerical solvers that rely on the cubic interpolated propagation (CIP) scheme. This method was firstly introduced in (Yabe, Ishikawa et al. 1991) as a universal solver for hyperbolic equation by cubic interpolation. It is then used for solving fluid dynamic equation as a conservative semi-Lagrangian solver for a solid, liquid and gas. Some studies prove that this method is efficient enough to solve hydrodynamic equation in all states (Yabe, Ogata et al. 2002).

More advance CIP study (Song, Shin et al. 2005) attempts to improve the *stable fluid* (Stam 1999) from numerical dissipation. This scheme is not just applicable for dissipative media (like fog and smoke), but also non-dissipative liquid (water). While, the CIP produce low dispersion compare to other method (Yabe, Takizawa et al. 2007), it need no dimensional splitting (Kim, Song et al. 2008). These provide user to simulate fluid in all state simultaneously. Lastly, although the CIP has been accepted as an alternative method for fluid dynamic solver that fits the need of computer animations, to the best of author knowledge, there is no other study to date that uses this method in surgery simulation including force feedback interaction device.

In this paper, we introduce our approach in combining CIP fluid solver and parallelism in GPU. The rest of the paper is organized as follows: Force model computation is described in Section III; GPU implementation for visual rendering and fluid solver computation is described in Section IV; Section V describes the experimental setups and the results; and lastly, Section VI describes a conclusion of the study and planned work for the future.



### 3 Force models computation

A *force model* is defined as a force generating feedback to haptic devices or other kinaesthetic feedback tools in a virtual environment. This force has a means of mapping from the haptic device position to a force vector (Anonymous 2005). There are techniques for integrating force models into virtual environment. This paper concentrates on integrating force as the existing fluid in the simulation.

#### 3.1 Fluid solver formulation

Theoretically, fluid dynamic equations are described by Navier-Stokes equations (NSEs). In compressible Newtonian fluids, the motion equations are derived from a combination of the transport of a *momentum* in the fluid media;

$$\frac{\partial u}{\partial t} = -(u \cdot \nabla)u - \frac{1}{\rho} \nabla p + \frac{\mu}{\rho} \nabla^2 u + F_c \quad (1)$$

and mass conservation function;

$$\nabla \cdot u = 0 \quad (2)$$

where  $u$ ,  $p$ ,  $\rho$  are a velocity field of fluid, a pressure, and the density of the fluid respectively. Whereas  $\mu$  is a viscosity coefficient,  $F_c$  represents external force (the gravity, forces delivered through haptic), and  $\nabla$  is the differential operator, respectively. The fluid movement can be predicted by solving the Equations (1) and (2). The flow simulation can be run in staggered grids that define velocity components at cell faces and scalar variables in cell centres.

In addition, If  $\phi$  is the function of the volume of fluid fraction, the fraction of fluid volume in each cell is transported by using advection equation

$$\frac{\partial \phi}{\partial t} = -(u \cdot \nabla) \phi \quad (3)$$

The solver of this equation can be generated d by using a combination of the CIP method and the advection form  $-(u \cdot \nabla)u$ . Then, the interface between liquid and solid is traced by distributing Equations (4) into advection phase (Eq.6) and non-advection (Eq.5). If  $g = -(u \cdot \nabla) f$ , then we can formulate that:

$$\frac{\partial \phi}{\partial t} + u \cdot \nabla \phi = g \quad (4)$$

$$\frac{\partial \phi}{\partial t} = g \quad (5)$$

$$\frac{\partial \phi}{\partial t} + u \cdot \nabla \phi = 0 \quad (6)$$

The solution of those equations in the spatial domain  $(x, y, z)$  can be performed by solving the non-advective part (Eq.5) using a *finite difference* and then advect the result as shown in Eq.6. Numerical solution for this, on the other hand, can be calculated in a grid of cells of the space domain.

Furthermore, the advection phase can be computed by shifting a cubic interpolated profile into space according to the total derivative equations. Eq.6 is also called as *level set equations*, where the surface of liquid can be

obtained by tracking their positions for which  $\phi = 0$ .

Then the exact solution of Eq.6 can be formulated as

$$\phi(x, t) = \phi(x - ut, 0) \quad (7)$$

If the velocity is assumed to have a constant value within a short time, we get

$$\phi(x, t + \Delta t) \cong \phi(x - u\Delta t, t) \quad (8)$$

Because of the CIP method uses not only the function values at the grid points, the spatial derivatives (Eq.5) at those positions then are used for constructing the profile inside the grid cell to trace  $\phi$  and  $g$ . The values  $\phi$  and  $g$  at the two grid points is interpolated by a cubic polynomial. Thus, we can obtain the profile at the next time step  $n+1$  by transporting  $u\Delta t$  and we have;

$$\phi_i^{n+1} = F(x_i - u_i \Delta t) = a_i \xi^3 + b_i \xi^2 + g_i^n \xi + \phi_i^n \quad (9)$$

$$g_i^{n+1} = dF(x_i - u_i \Delta t) = 3a_i \xi^2 + 2b_i \xi + g_i^n$$

where

$$a_i = \frac{g_i + g_{i'}}{D^2} + \frac{2(\phi_i - \phi_{i'})}{D^3}$$

$$a_i = \frac{3(\phi_{i'} - \phi_i)}{D^2} - \frac{2g_i + g_{i'}}{D}$$

and  $\xi = -u\Delta t$ , when  $u > 0$  we have  $D = -\Delta x$ ,  $i' = i-1$  otherwise when  $u < 0$ ,  $D = \Delta x$ ,  $i' = i+1$ . However the use of cubic profile (Eq.9) can lead to instabilities. The modification as proposed in (Song, Shin et al. 2005) solve this problem and is claimed always stable. This scheme implement higher dimensional CIPs based on the monotonic CIP solver.

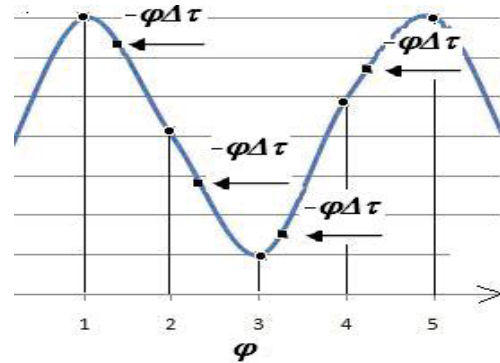


Figure 1. The physical value estimation in advection phase on the departure point of the arbitrary contour.

The numerical solver in CIP scheme employs a spatial profile that rely on different interpolations to propagate the solution along the characteristics (Yabe, Mizoe et al. 2004). The physical value on  $\phi_i$  is calculated locally within the computed cell (the original equation). Then, an estimated value is directly advected toward the grid point calculations at the next time step value (Fig.1). This step provides results in more accurate modelling of the real situation, even in fairly coarse grid. While other methods require multi-points to construct the profile in the one dimensional case, the CIP can be constructed only from a single cell. This advantage is useful for treating boundaries. Beside the CIP method provides a stable result, less diffusive, and has third order accuracy (Utsumi, Kunugi et al. 1997; Song, Shin et al. 2005); the numerical solver of fluid motions provide the solutions

simultaneously to all interfaces in all states without dimensional splitting (Kim, Song et al. 2008).

### 3.2 Interaction forces

The external forces or forces driven by user interaction,  $F_c$  in (Eq.1) can consist of gravity ( $F_g$ ), surface tension ( $f_t$ ), a user interactive force (haptic) ( $F_u$ ), or a combination of them. These forces can be represented as below:

$$F_c = F_g + f_t + F_u$$

When the surface tension is treated as a body force and no explicit information on the geometry and position of the liquid surface is required, the surface tension can be expressed as a continuum surface force as follow (Brackbill, Kothe et al. 1992):

$$f_t = -\sigma k(\phi) \delta_\epsilon(\phi) \nabla(\phi)$$

where  $\sigma$  and  $k(\phi)$  are surface constant coefficient and local curvature value respectively, and  $\delta(\phi)$  delta function that is formulated as:

$$\delta_\epsilon(\phi) = \begin{cases} \frac{1}{2\epsilon} (1 + \cos(\frac{\pi\phi}{\epsilon})) & |\phi| \leq \epsilon \\ 0 & |\phi| > \epsilon \end{cases}$$

Interactive force from user through a haptic device can be formulated as the equation of motion for a rigid body (Eq.10).

$$m\ddot{x} = g - \int p n ds \quad (10)$$

where  $m$  is the mass and  $x$  the centre of gravity of the rigid body. (Eq.1) requires the solver for the fluid pressure or commonly called as pressure projection. This projection is solved using the Poisson's equation with assumption that the liquid has no viscosity and has an external force that is determined by the force released from/to haptic interactions. After the fraction of the volume of fluid in each cell is obtained in the simulation, a fluid surface is created and it is smoothened by subdivision, if necessary. Assuming that  $u'$  is the velocity result obtained by processing (Eq.1), the pressure can be expressed as the Poisson equation

$$\nabla \cdot \left( \frac{\nabla p}{\rho} \right) = \frac{\nabla \cdot u'}{\Delta t} \quad (11)$$

(Eq.11) can be discretised as:

$$\sum_{\psi=(i,j,k)} (\rho_{\psi+\frac{1}{2}}^{-1} + \rho_{\psi-\frac{1}{2}}^{-1}) p_\psi - (\rho_{\psi+\frac{1}{2}}^{-1} p_{\psi+1} + \rho_{\psi-\frac{1}{2}}^{-1} p_{\psi-1}) = -\frac{1}{\Delta t} \sum_{\psi=(i,j,k)} (u'_{\psi+\frac{1}{2}} - u'_{\psi-\frac{1}{2}})$$

This equation reveals that velocity and density values can be taken from cell faces, whereas the pressure values are taken from the centres of neighbouring cells, as shown in Fig.2.

### 3.3 Repelling forces

A force that is received from or released to the finger through haptic device is called as *repelling forces*. This force can have the following expression:

$$d\phi = -np(\phi)ds \quad (12)$$

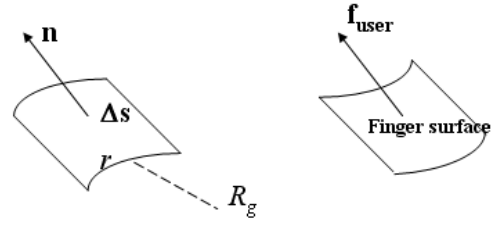


Figure 2. Field element of fluid (left) and force acting on the user (right).

where  $\phi$  is the position object that may be not located in the grid cell, hence pressure calculation need to be interpolated based on the CIP scheme. The determination of pressure is critical to get precise repelling force and we need an accurate profile of the pressure inside a grid cell. The pressure may leads to the rigid body torque ( $\Gamma$ ). The formulation of the torque can be given as:

$$d\Gamma = -np(\phi) \times (\phi - r) ds \quad (13)$$

These values are updated in each simulation step by running computation kernels over the grid. The computation is implemented as a vertex shader that executes on every cell in the grid and the results are written to an output texture. Since GPUs are designed to render into 2D buffers, the execution must be done for each slice of a 3D volume by iterating slice indexes over the entire grid.

The momentum equation is calculated numerically by splitting the equations into advection and non-advection equation groups. The advection equations consist of velocity, density, and level set advectons, whereas non advection equations have pressure projection and diffusions.

## 4 Experimental Setup

Data collected in this paper was generated from objects that were digitally created from a human anatomical test sample. The test results demonstrated the efficiency and effectiveness of our proposed framework for a 3D fluid dynamic simulation in real time interactions with force feedbacks. All results were implemented on a personal computer (PIV Dual Core 3.2 GHz, 2GB RAM, 80GB HD) in combination with the graphics chip of a NVIDIA GeForce 8800 GTX 758MB run on 400 MHz RAMDAC Clock Speed. To support the immersion and kinaesthetic felling, the PHANTOM Haptic premium 1.5 (from SensAble) in Reachin display were used.

The visual and haptic rendering were developed based on Reachin API 4.2 and blended in Visual Studio.net, Python 2.4, and VRML. Mostly the computations in this study were implemented with 32-bit floating points on programmable graphics hardware that was assumed suitable to real-world problems.

Flow charts of the simulation system in the experiment can be described as in Fig 3. The system is divided into two main threads of subsystem: slow and fast subsystem. Slow subsystem manages visual rendering with frequency of about 20-60 Hz, whereas fast subsystem controls haptic rendering in the frequency of about 1000Hz.

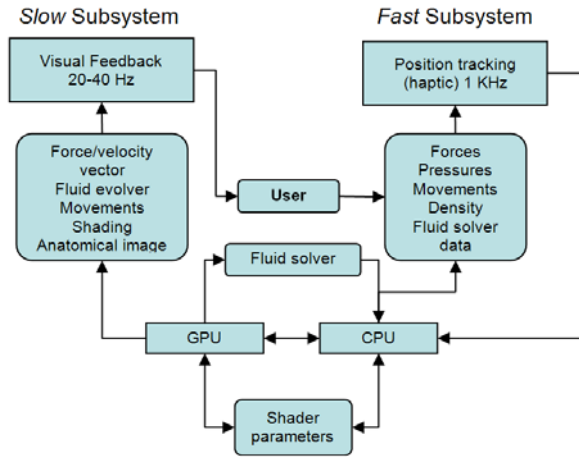


Figure 3. Flow chart CPU-GPU balancing for fluid dynamic visualisation in heart beating surgery simulation.

## 5 Results and Discussion

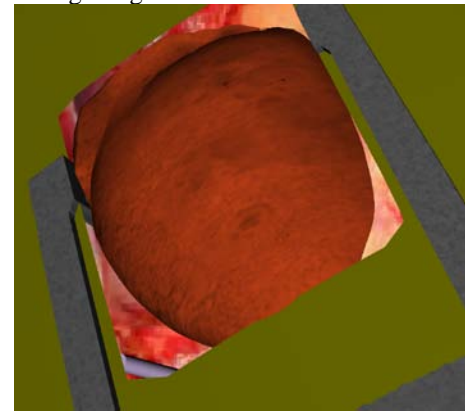
This study examines fluid-haptic interactions in several different scenarios of heart surgery simulations. The medical data in this paper were digitally created from a human anatomical test sample (Anonymous 2003).

The average performance observed during the experiment can be seen in the Table 1. The data is observed based the frame rate on of  $V_{fps}$  (Visual frame rate per second) and  $H_{fps}$  (Haptic frame rate per second). The comparison and contrast of the surgery simulation with and without GPU computations are presented. CPU computation means that all the calculation of the solver were undertaken in CPU and the GPU was only for visualization. CPU-GPU computation means that the computations were done on GPU with CPU as a host controller to manage computation and visualization on the GPU.

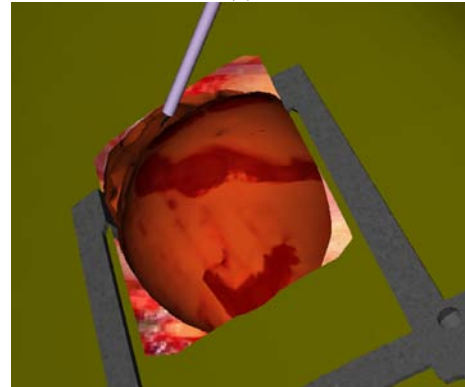
Type of Interaction	CPU Computation (average)		CPU-GPU Computation (average)	
	( $V_{fps}$ )	( $H_{fps}$ )	( $V_{fps}$ )	( $H_{fps}$ )
Cuttings with no bleeding	22	1001	49	1001
Blood flowing test without Cutting (Injector)	24	980	54	1000
Cutting and bleeding	10	960	43	1001
Cutting -bleeding in slow motion	8	940	29	993
Cutting -bleeding in fast motion	4	920	25	992
Cutting -bleeding in fast motion (short)	4	920	23	955
Cutting -bleeding in fast motion (Long)	2	900	18	953

Table 1: Frame rate CPU versus GPU computations

In Table 1, a *cutting* represents a combination of the pressure force and the sharpness constant of the blade. *Cutting without bleeding* represent the voxel removal along the cutting path. When it is *combined with bleeding*, the computation will be distributed into two tread groups, one group for managing voxel removal and the other for driving the fluid solvers. *Slow or fast movement*, on the other hand, influences the speeding of voxel removal computations and the releasing fluid from the cutting path. When cutting force is applied, the fluid will be released from the cut position and will continue to flow through the heart surface toward the boundary or towards the lowest parts on the surface and toward the centre of the gravity force. An example of snapshot simulation is shown in Figure 4a and 4b. This example describes the cutting simulation followed by blood flowing over the surface in the cutting path during heart beating surgery. It seems that as the faster scalpel move the frame rate getting slower.



(a)



(b)

Figure 4. Snapshot heart beating surgery. (a) preparation with retractor to keep the surface open. (b) Blood flowing over the heart surface.

Using GPU can increase the performance of the simulation. From Table 1, the performance increase about 2.2 to over 12 times when GPU computation is implemented on the simulation. The more complex the computation, the better the performance with GPU. It may reveal that when the computations need more threads, using GPU provides more advantage.

## 6 Conclusion

This paper presents a fluid dynamic modelling for heart surgical simulator. Our work focus on the development of the real-time fluid dynamic with haptic interaction

based on the CPU-GPU balancing to get more realism fluid effects in heart beating surgery simulator. By taking advantage of the GPU parallelism coupling with CIP fluid solver that solve multiphase fluid simultaneously, the fluid dynamic simulation can be created in real time without involving supercomputer.

Our results demonstrated the great potential for designing plausible surgical simulator with complex fluid effects in real-time, particularly for virtual reality simulator with kinaesthetically interaction. It is demonstrated that the basic need of fluid effects in surgical simulation can be created such as: cutting and bleeding, fluid injection, and fluid flowing over complex obstacles. They can be effectively simulated on realistic visualization on reasonable frame rate.

Future attempts in improving virtual surgical simulation include multi-fluid interactions, surgical tools simulations (such as: sucker and injector), and a complex solid-fluid interactions (such as mixing blood and other surgical fluids).

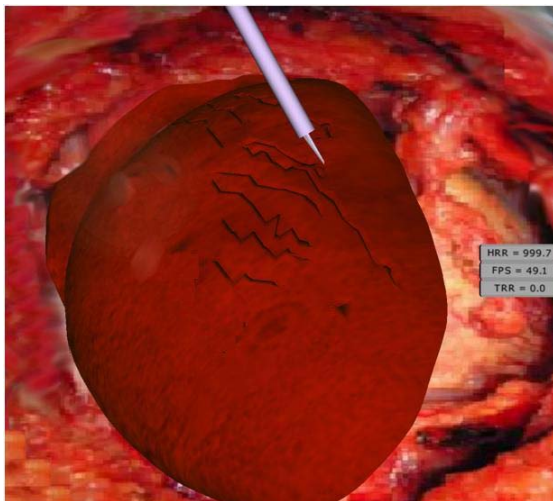
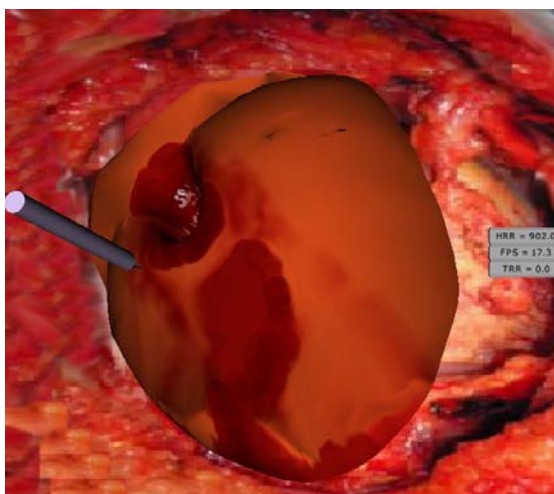
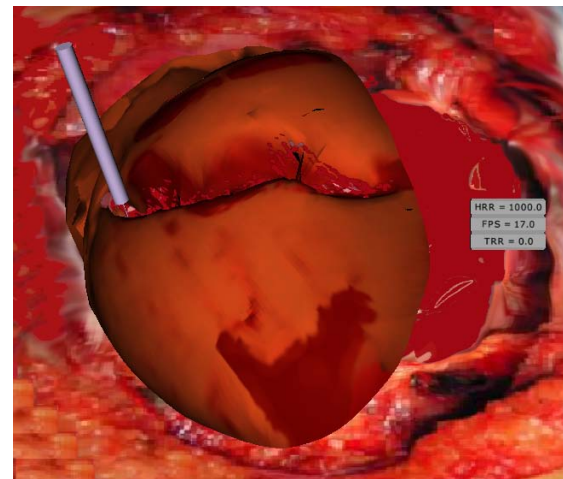


Figure 5. Snapshot of cutting without bleeding (Voxel cutting over the surface of the heart).



(a)



(b)

Figure 6. Combination cutting-opening-bleeding during heart beating surgery simulations: (a) short movement), (b) long movement.

## 7 References

- Adrien, T., L. Andrew, et al. (2006). "Model reduction for real-time fluids." *ACM Trans. Graph.* **25**(3): 826-834.
- Anh, N. H., A. Sourin, et al. (2007). Physically based hydraulic erosion simulation on graphics processing unit. *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*. Perth, Australia, ACM New York, NY, USA.
- Anonymous. (2001). "Virtual Simulation of Temporal Bone Dissection." from <http://www.osc.edu/Biomed/temporal.html> [4/5/2001].
- Anonymous (2003). The Visible Human Project. U. S. N. L. o. Medicine, [http://www.nlm.nih.gov/research/visible/visible\\_human.html](http://www.nlm.nih.gov/research/visible/visible_human.html).
- Anonymous (2005). ReachinAPI 4 Programmer's Guide, Reachin Technology AB.
- Avila, R. S. and L. M. Sobierajski (1996). *A haptic interaction method for volume visualization*. Visualization '96. Proceedings.
- Baxter, W. and M. C. Lin (2004). Haptic interaction with fluid media. *Proceedings of the 2004 conference on Graphics interface*. London, Ontario, Canada, Canadian Human-Computer Communications Society: 81-88.
- Benjamin, L. and R. Erik (2009). Real-time fluid simulation using discrete sine/cosine transforms. *Proceedings of the 2009 symposium on Interactive 3D graphics and games*. Boston, Massachusetts, ACM.
- Bhasin, Y., A. Liu, et al. (2005). "Simulating Surgical Incisions Without Polygon Subdivision." *IOS Press* **111**: 43-49.
- Brackbill, J. U., D. B. Kothe, et al. (1992). "A continuum method for modeling surface tension." *J. Comput. Phys.* **100**(2): 335-354.
- Csebfalvi, B. and S.-K. Szirmay-Kalos (2003). Monte Carlo Volume Rendering. *Proceedings of the 14th*



- IEEE Visualization 2003 (VIS'03), IEEE Computer Society: 59.
- Foster, N. and D. Metaxas (1996). "Controlling Realistic Animation."
- Higgins, A. and S. Koucky (2002). "Mission impossible?" Machine Design **74**(23): 28.
- Kass, M. and G. Miller (1990). Rapid, stable fluid dynamics for computer graphics. Proceedings of the 17th annual conference on Computer graphics and interactive techniques, Dallas, TX, USA, ACM.
- Kim, D., O.-y. Song, et al. (2008). "A Semi-Lagrangian CIP Fluid Solver without Dimensional Splitting." Computer Graphics Forum (Proc. Eurographics) **27**(2): 467-475.
- Kim, L. and S. H. Park (2004). A Haptic Sculpting Technique Based on Volumetric Representation. Lecture Notes in Computer Science, F. J. Perales and B. A. Draper, Springer-Verlag. **3179/2004**: 14-25.
- Kruger, J., P. Kipfer, et al. (2005). "A particle system for interactive visualization of 3D flows." Visualization and Computer Graphics, IEEE Transactions on **11**(6): 744-756.
- Losasso, F., G. Irving, et al. (2006). "Melting and Burning Solids into Liquids and Gases." Visualization and Computer Graphics, IEEE Transactions on **12**(3): 343-352.
- Lundin, K., B. Gudmundsson, et al. (2005). General proxy-based haptics for volume visualization. First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. WHC 2005.
- Lundin, K., M. Sillen, et al. (2005). Haptic Visualization of Computational Fluid Dynamics Data Using Reactive Forces, the Conference on Visualization and Data Analysis, part of IS&T/SPIE Symposium on Electronic Imaging San Jose, CA USA.
- Matthias, M. and Iler (2009). Fast and robust tracking of fluid surfaces. Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation. New Orleans, Louisiana, ACM.
- Mei, X., P. Decaudin, et al. (2007). Fast Hydraulic Erosion Simulation and Visualization on GPU. Computer Graphics and Applications, 2007. PG '07. 15th Pacific Conference on.
- Muller, M., B. Solenthaler, et al. (2005). Particle-based fluid-fluid interaction. Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation. Los Angeles, California, ACM Press: 237-244.
- Noe, K. O. and P. Trier (2004). Implementing Rapid, Stable Fluid Dynamics on the GPU.
- Owens, J. D., D. Luebke, et al. (2005). "A Survey of General-Purpose Computation on Graphics Hardware" Eurographics 2005, State of the Art Reports(August): 30.
- Song, O.-Y., H. Shin, et al. (2005). "Stable but nondissipative water." ACM Trans. Graph. **24**(1): 81-97.
- Stam, J. (1999). "Stable Fluids." Proc. of the 26th Annual Conference on Computer Graphics and Interactive Techniques: 121-128.
- Stam, J. and E. Fiume (1993). "Turbulent Wind Fields for Gaseous Phenomena." Computer Graphics Proceedings: 369-372.
- Stanbridge, R. D. L., David O'Regan, et al. (1999). "Use of a Pulsatile Beating Heart Model for Training Surgeons in Beating Heart Surgery." The Heart Surgery Forum **2**(4): 4.
- Stone, R. J. (2000). "Haptic feedback: A potted history, from telepresence to virtual reality." Lecturer notes in computer science **2058/2001**(August/September): 1-7.
- Thomas V, T. I. and E. Cohen (1999). Direct Haptic Rendering Of Complex Trimmed Nurbs Models. Proceedings of Symposium on Haptic Interfaces, Providence, ASME International Congress and Exposition.
- Utsumi, T., T. Kunugi, et al. (1997). "Stability and accuracy of the Cubic Interpolated Propagation scheme." Computer Physics Communications **101**(1-2): 9-20.
- Yabe, T., T. Ishikawa, et al. (1991). "A universal solver for hyperbolic equations by cubic-polynomial interpolation II. Two- and three-dimensional solvers." Computer Physics Communications **66**(2-3): 233-242.
- Yabe, T., H. Mizoe, et al. (2004). "Higher-order schemes with CIP method and adaptive Soroban grid towards mesh-free scheme." Journal of Computational Physics **194**(1): 57-77.
- Yabe, T., Y. Ogata, et al. (2002). "The next generation CIP as a conservative semi-Lagrangian solver for solid, liquid and gas." Journal of Computational and Applied Mathematics **149**(1): 267-277.
- Yabe, T., K. Takizawa, et al. (2007). "Computation of fluid-solid and fluid-fluid interfaces with the CIP method based on adaptive Soroban grids - An overview." International Journal for Numerical Methods in Fluids **54**(6-8): 841-853.



# Improved Consensus Clustering via Linear Programming

Nicholas Downing<sup>1</sup>Peter J. Stuckey<sup>1,2</sup>Anthony Wirth<sup>1</sup>

<sup>1</sup>Department of Computer Science and Software Engineering  
University of Melbourne  
Victoria, 3010  
Australia

<sup>2</sup> National ICT Australia, Victoria Laboratory  
Email: {ndowning@students., pjs@, awirth@}csse.unimelb.edu.au

## Abstract

We consider the problem of CONSENSUS CLUSTERING. Given a finite set of input clusterings over some data items, a *consensus* clustering is a partitioning of the items which matches as closely as possible the given input clusterings. The best exact approach to tackling this problem is by modelling it as a Boolean Integer Program (BIP). Unfortunately, the size of the BIP grows cubically in the number of data items, hence this method is applicable to only small sets of items. In this paper we show how to tackle the problem progressively, leading to much improved solution times and far less memory usage than previously. For the case where approximate clusterings are acceptable, we show a number of heuristic techniques for extracting good clusterings from the solutions of the linear relaxation of the BIP, and on several very large data sets we demonstrate much higher quality approximations than previously possible. When optimal solutions are desired, the problem is much harder, and we present some novel and existing techniques that can assist in finding candidate answers and proving the optimality thereof. For the first time we present optimal Consensus Clusterings for several complete, albeit small, data sets.

**Keywords:** Clustering, Linear Programming, Integer Programming, Combinatorial Optimization.

## 1 Introduction

We consider the CONSENSUS CLUSTERING problem. Given a population of items, we can define a *cluster* to be a subset of those items, and a *clustering* to be a set of clusters such that each item falls into exactly one cluster. Finding the *consensus* means generating an output clustering which closely matches some ( $> 1$ ) given input clusterings, over a common population. When the given input clusterings don't agree exactly, we must compromise to find a consensus that is sufficiently close to all of them.

Defining this formally we need the notion of a distance between clusterings. There are several sensible measures of distance between clusterings, but we restrict our attention to the *Mirkin metric* (Mirkin & Chernyi 1970), also proposed by Filkov & Skiena (2003) who base their work on Rand (1971) and refer to the metric as the *unnormalized Rand distance*. Suppose we are given a clustering  $\pi_x$  described by an

Copyright ©2010, Australian Computer Society, Inc. This paper appeared at the Thirty-Third Australasian Computer Science Conference (ACSC2010), Brisbane, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 102, B. Mans and M. Reynolds, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

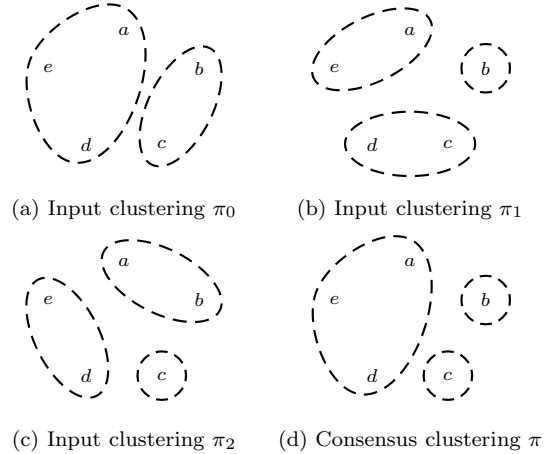


Figure 1: Example of a consensus between clusterings

adjacency  $x_{uv}$  such that

$$x_{uv} = \begin{cases} 0, & \text{items } u, v \text{ are in the same cluster} \\ 1, & \text{items } u, v \text{ are in different clusters,} \end{cases}$$

and similarly  $\pi_y$  defined by  $y_{uv}$ , over a common population of items  $V$ , then the Mirkin metric  $d(\pi_x, \pi_y)$  can be defined as one half the number of ordered pairs  $(u, v) \in V^2$  for which  $x_{uv} \neq y_{uv}$ ; the factor of one half is because  $(u, v)$  and  $(v, u)$  make the same symmetric contribution.

Since  $d$  is a metric, clusterings form a metric space and the *consensus* should be a kind of median or centroid of the given input clusterings. We use Filkov and Skiena's (2003) definition of CONSENSUS CLUSTERING: Given  $m$  partitions  $\pi_0, \pi_1, \dots, \pi_{m-1}$ , find a consensus partition  $\pi$  that minimizes

$$S = \sum_{k=0}^{m-1} d(\pi_k, \pi). \quad (1)$$

Figure 1 illustrates an example where  $\pi$  is the consensus between clusterings  $\pi_0, \pi_1, \pi_2$ . In this example,

$$\begin{aligned} d(\pi_0, \pi) &= 1 && \text{because of the } (b, c) \text{ disagreement;} \\ d(\pi_1, \pi) &= 3 && \text{because of } (a, d), (c, d) \text{ and } (d, e); \\ d(\pi_2, \pi) &= 3 && \text{because of } (a, b), (a, d) \text{ and } (a, e). \end{aligned}$$

Therefore,  $S = 7$  for the consensus shown. That  $S$  is minimal seems plausible since the output clustering shares some features of each input clustering. For a small instance like this, it is feasible to enumerate all possible solutions in order to convince oneself that  $S$  is minimal. The problem in general is NP-complete (Wakabayashi 1998).

Earlier work on CONSENSUS CLUSTERING focuses on the distance function between clusterings, motivated mainly by the desire for a general purpose way to measure the performance of other (specific purpose) distance functions and clustering methods (Rand 1971).

Aside from its interestingness as a difficult combinatorial optimization problem, CONSENSUS CLUSTERING is quite widely applicable. Filkov & Skiena (2003) have applied CONSENSUS CLUSTERING to microarray gene expression data. Gionis et al. (2007) are the most lucid in describing some of the applications,

- clustering of categorical data;
- clustering of heterogeneous data;
- identifying the correct number of clusters;
- detecting outliers;
- improving clustering robustness;
- privacy-preserving clustering.

For the experiments described in this paper, we consider three data sets of the categorical type (POSTOPERATIVE, LYMPHOGRAPHY, and MUSHROOMS) and one of the heterogeneous type (ADULT), available from the UCI Machine Learning repository (Asuncion & Newman 2007).

CONSENSUS CLUSTERING is an example of clustering based on inconsistent advice. A more general example of inconsistent advice is when we are given a graph with labelled edges such as Figure 2. The edge labels are the ‘advice’, where + requests that a pair of items (vertices) be placed in the same cluster and – requests that they be placed in different clusters. We then aim to cluster them so that as few – edges span clusters and as many + edges are within clusters as possible.

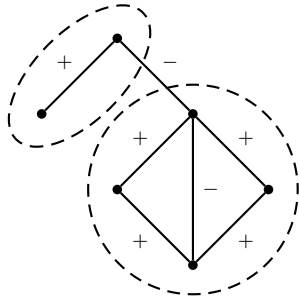


Figure 2: Example of inconsistent advice

In this example, a ‘perfect’ clustering is not possible due to the inconsistency of the advice, but the clustering shown is optimal in the sense that one of the – edges can be respected whereas the other – edge should not be respected as it would involve violating at least two + edges.

This example follows Coleman et al. (2008) who refer to the problem as 2-CORRELATION CLUSTERING. The motivation is that in biological experiments (protein or gene clustering) it may be practical to test associations between pairs, keeping in mind that experimental error could cause inconsistency in the advice. Naturally then, we should try to respect as much of the advice as possible.

The problem described forms part of the CORRELATION CLUSTERING problem-family. The paper which introduced CORRELATION CLUSTERING to the theory community was by Bansal et al. (2002). The motivation for the original paper is a document clustering problem in which one has a pairwise similarity function  $f$  learned from past data, and the goal is to partition documents in a way that correlates with  $f$  as much as possible.

Defining the problem formally it is helpful to consider a complete graph with real-valued edge weights. Suppose in the above example,  $V$  is the set of vertices, – edges are relabelled  $(0, 1)$  and + edges are relabelled  $(1, 0)$ . Then each edge label  $(w_{uv}^-, w_{uv}^+)$  consists of two parts, the penalty  $w_{uv}^-$  for clustering items  $u, v \in V$  together, and the penalty  $w_{uv}^+$  for clustering them apart.

The graph can easily be made complete by introducing edges labelled  $(0, 0)$ , and the problem becomes that of finding the clustering  $\pi_x$  and its adjacency  $x_{uv}$  that minimizes

$$S = \frac{1}{2} \left( \sum_{x_{uv}=0} w_{uv}^- + \sum_{x_{uv}=1} w_{uv}^+ \right), \quad (2)$$

where  $(u, v) \in V^2$  and the factor of one half is again because  $(u, v)$  and  $(v, u)$  make the same symmetric contribution.

The taxonomy of Emanuel & Fiat (2003) considers 3 different versions of the problem,

- unweighted complete graph: Bansal et al. (2002);
- unweighted general graph: Figure 2;
- weighted general graph: Equation (2);

listed in increasing order, that is, each set of problems is a subset of the next. Their motivation is a set of clustering problems described by Cohen & Richman (2001, 2002), in which learning algorithms were trained to help with various clustering tasks e.g. clustering of entity names to determine whether names from different databases refer to the same person.

The 3rd formulation, which we use, has another important application. The CONSENSUS CLUSTERING problem, as defined previously, can be reduced to CORRELATION CLUSTERING of a weighted graph. It is simply a matter of considering the contribution of an individual pair  $(u, v) \in V^2$  to the objective function  $S$  over all input clusterings  $\pi_0, \pi_1, \dots, \pi_{m-1}$ . Let  $y_{uv}^k$  be the adjacency of  $\pi_k$ , and let

$$\begin{aligned} w_{uv}^- &= \text{how many } \pi_k \text{ have } y_{uv}^k = 1 \\ w_{uv}^+ &= \text{how many } \pi_k \text{ have } y_{uv}^k = 0. \end{aligned}$$

Referring to Equation (2), it is clear that  $S$  counts how many  $x_{uv} \neq y_{uv}^k$ , as is required by the original CONSENSUS CLUSTERING definition.

It is generally more convenient to tackle CONSENSUS CLUSTERING via the CORRELATION CLUSTERING interpretation rather than directly considering distances between clusterings, but we take some shortcuts e.g. we know that  $w_{uv}^+ + w_{uv}^- = m$  (the so-called *probability condition*), which happens to be true for CONSENSUS CLUSTERING but isn’t a general requirement for CORRELATION CLUSTERING.

## 2 Formulating the Linear Program

Considering numbered items  $0, 1, \dots, s-1$  and referring to Equation (2), we want to find the adjacency  $x_{ij}$  that minimizes

$$S = \sum_{0 \leq i < j < s} ((1 - x_{ij})w_{ij}^- + x_{ij}w_{ij}^+).$$

The constraint on this minimization is that  $\mathbf{x}$  must represent a clustering  $\pi$ , which means  $x_{ij} \in \{0, 1\}$ , and  $\{(i, j) : x_{ij} = 0\}$  must be an equivalence relation. We use the notation  $x_{ij}$  a little loosely, to mean one of  $x_{ij}$ , 0, or  $x_{ji}$  when  $i < j$ ,  $i = j$ , or  $i > j$  respectively, hence symmetry and reflexivity are guaranteed.



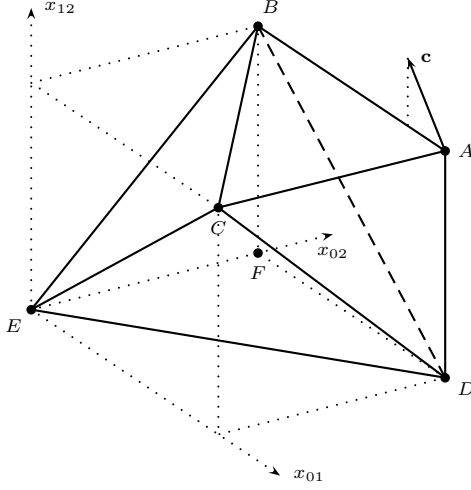


Figure 3: Solution space of a 3-item problem

Transitivity is naturally expressed by considering the  $x_{ij}$  as ‘distances’ subject to the usual *triangle inequality* constraints which specify that there should be no ‘shortcut’ between items  $(i, j)$  via some other item  $k$ . Expressing the problem in this way yields the Boolean Integer Program (BIP)

$$\begin{aligned} \min. \quad & S = \sum_{0 \leq i < j < s} (w_{ij}^+ - w_{ij}^-)x_{ij} + \text{constant} \\ \text{s.t.} \quad & x_{ij} \in \{0, 1\} \\ & x_{ij} \leq x_{ik} + x_{kj} \quad \forall i, j, k, \end{aligned}$$

as proposed by earlier authors (Charikar et al. 2003, Ailon et al. 2005). For practical reasons we relax the integrality constraints giving the final LP

$$\begin{aligned} \min. \quad & S = \sum_{0 \leq i < j < s} (w_{ij}^+ - w_{ij}^-)x_{ij} \\ \text{s.t.} \quad & 0 \leq x_{ij} \leq 1 \\ & x_{ij} \leq x_{ik} + x_{kj} \quad \forall i, j, k. \end{aligned} \quad (3)$$

We return to the issue of recovering a BIP solution from the LP solution in Sections 4 and 7.

We can solve this LP with the Simplex method. For a simple 3-item example, this can be illustrated visually. Figure 3 shows the effect of the constraints in the 3-dimensional cube formed by the coordinates  $x_{01}, x_{02}, x_{12}$ . The feasible region, called the Simplex, is outlined by the solid and dashed lines, consisting of those points in the 3-d solution space which satisfy all constraints.

Each vertex  $A, B, C, D, E$  of the Simplex is necessarily at the intersection of  $n$  constraint boundaries, where  $n = (s - 1)s/2$  is the dimension of the space, here 3. For example, making the constraints  $x_{01} \leq 1$ ,  $x_{02} \leq 1$ , and  $x_{12} \leq 1$  tight (i.e. equalities) gives us the vertex  $A$ . Edges traverse the intersection of only 2 constraint boundaries, e.g. making the constraint  $x_{01} \leq 1$  no longer tight, gives us the edge  $\vec{AB}$ .

We show a hypothetical objective vector  $\mathbf{c}$  in the direction of steepest increase (worsening) of  $S$ . Since this vector has a negative component in the direction  $\vec{AD}$ , moving from  $A$  to  $D$  improves the solution. Each constraint then has a *shadow price* which is the component of the objective vector in the direction we would move if the constraint were loosened. The tight constraints which intersect to form the current solution are called *non-basic*, the remaining, possibly

```

procedure AllViolated
   $R \leftarrow \emptyset$ 
  repeat
     $\mathbf{x}^* \leftarrow \text{argmin}\{S(\mathbf{x}) : \mathbf{x} \text{ satisfies } R\}$ 
     $\text{violations} \leftarrow \text{false}$ 
    for each  $0 \leq i < j < s, 0 \leq k < s, k \notin \{i, j\}$ 
    do
      if  $x_{ij}^* > x_{ik}^* + x_{kj}^*$  then
         $R \leftarrow R \cup \{x_{ij} \leq x_{ik} + x_{kj}\}$ 
         $\text{violations} \leftarrow \text{true}$ 
      end if
    end for
  until  $\text{violations} = \text{false}$ 
end procedure

```

Algorithm 1: Entering only the violated constraints

loose, constraints are *basic*. *Pivoting* means moving along an edge between vertices, that is, swapping the basic status of some non-basic constraint with that of the appropriate basic constraint.

### 3 Solving the LP

The resulting LP (3) requires  $(s - 1)s/2$  variables and  $(s - 2)(s - 1)s/2$  constraints (omitting the trivial constraints that arise when some of  $i, j$ , and  $k$  are identical). We can enter this directly into the solver (procedure ALLCONSTRAINTS), but there is a substantial amount of redundancy in the LP. Instead of trying to solve the whole problem initially, we can tackle it progressively.

The simplest approach is just to begin with the solution to a relaxed problem having no triangle inequality constraints, and progressively add the necessary constraints whenever a violated triangle inequality is found in the proposed solution. Each iteration, we add constraints for all violations and then re-solve to optimality, as shown in Algorithm 1. The solver is warm-started after adding new constraints.

The checking for violations as shown is a brute force algorithm, involving 3 nested loops. In practice, if for  $(i, j)$  we have  $x_{ij}^* = 0$ , we can avoid looping over  $k$ . Furthermore, we can skip many  $k$ -ranges by pre-computing the sets of  $k$ -values  $K_i, K_j$  for which the RHS might be  $< 1$  and intersecting them in the inner loop. Since this involves only a few instructions, the checking time was much less than the solving time, in our experiments. Hence we did not look at improving this brute force approach further.

#### 3.1 Choice of Simplex algorithm

Duality is one of the most important aspects of Linear Programming. We haven't space for a comprehensive introduction, but suffice it to say that we can construct an auxiliary LP which gives a lower bound on  $S$ , and maximizing this auxiliary LP not only finds the optimal  $S$  of the LP (3), but a solution to the original (*primal*) LP can be recovered from the shadow prices of the auxiliary (*dual*) LP, and vice versa.

The dual LP looks very much like the primal LP except that the constraint matrix is transposed and other roles are swapped. This is very useful when adding constraints to an LP that has already been solved to optimality, as in Algorithm 1 (ALLVIOLATED). The new constraints appear as variables in the dual, and variables can be added without cutting off the current solution. So if we restrict our attention to the dual, we can warm-start the solver much more easily after adding constraints each time. Hence we request that the solver utilize the *Dual Simplex* algo-

```

procedure SomeViolated
   $R \leftarrow \emptyset$ 
  repeat
     $\mathbf{x}^* \leftarrow \operatorname{argmin}\{S(\mathbf{x}) : \mathbf{x} \text{ satisfies } R\}$ 
     $V \leftarrow \emptyset$ 
    for each  $0 \leq i < j < s, 0 \leq k < s, k \notin \{i, j\}$ 
      do
        if  $x_{ij}^* > x_{ik}^* + x_{kj}^*$  then
           $V \leftarrow V \cup \{x_{ij} \leq x_{ik} + x_{kj}\}$ 
        end if
      end for
     $R \leftarrow R \cup \{\text{random subset of } V \text{ of size } \leq p\}$ 
  until  $V = \emptyset$ 
end procedure

```

Algorithm 2: Entering only a selection of the violated constraints

rithm, although in reality the dual might be formed in the presolving phase and *Primal Simplex* used.

Considering the use of the dual, we can now explain a subtlety in the formulation of the LP (3). Looking at the origin  $E$  of Figure 3, we notice that the constraints  $x_{ij} \geq 0$  are redundant in this instance. In general, the triangle inequalities imply that

$$x_{ij} \geq |x_{ik} - x_{kj}| \geq 0 \quad \forall i, j, k,$$

hence, when  $s \geq 3$  it would be valid to omit the bounds constraints  $x_{ij} \geq 0$ , in addition to the trivial triangle inequalities  $x_{ii} \leq x_{ij} + x_{ji}$  which are effectively the same thing. Then, making all the  $x_{ij} \leq 1$  bounds constraints tight, as at the vertex  $A$ , gives us a good *primal*-feasible starting solution.

On the other hand, double-bounded variables are very useful when considering the *dual* starting solution. Because the complementary constraints  $x_{ij} \geq 0$  and  $x_{ij} \leq 1$  have opposing shadow prices, we can construct a good dual-feasible starting solution by selecting the appropriate bound for each variable to make all shadow prices, and hence dual variables,  $\geq 0$ .

Removing the lower bound constraints significantly increases the solution time, showing that the solver in our experiments is smart enough to make use of the bounds constraints in this way. It is valid to remove all the bounds constraints not needed in the starting solution (including the upper bound on variables with strictly positive  $S$ -coefficients, as they have no reason to exceed their upper bound in any case), but this didn't significantly improve solution times, so we opted for the simplest possible formulation.

### 3.2 Entering only some violations

A further improvement on Algorithm 1 (ALLVIOLATED) is to add constraints for only a selection of the violations. In Algorithm 2 we randomly select  $p$  of the violated constraints to add. Informal experiments suggest that setting  $p$  to the number of variables in the problem, for a roughly square constraint-matrix (at least in the absence of degeneracy, which we discuss in Section 3.3), is a good heuristic.

The idea of choosing a selection was prompted by an informal experiment where we added all constraints, as in Algorithm 1, then performed a pivot and checked how many of the constraints remained violated. As we expected, if the pivot changed a variable  $x_{ij}$ , then a great many previously-violated constraints containing  $x_{ij}$  were no longer violated and hence of little further usefulness in improving the solution. Adding fewer constraints avoids many of these inter-dependencies, while achieving the aim of fixing violations and resolving conflicts in the advice.

```

procedure DeleteLoose
   $R \leftarrow \emptyset$ 
   $S_{prev} \leftarrow -\infty$ 
  repeat
     $\mathbf{x}^* \leftarrow \operatorname{argmin}\{S(\mathbf{x}) : \mathbf{x} \text{ satisfies } R\}$ 
    if  $S(\mathbf{x}^*) > S_{prev}$  then
       $S_{prev} \leftarrow S(\mathbf{x}^*)$ 
      for each  $r \in R$  do
        if the constraint  $r$  is basic then
           $R \leftarrow R \setminus \{r\}$ 
        end if
      end for
    end if
     $V \leftarrow \emptyset$ 
    for each  $0 \leq i < j < s, 0 \leq k < s, k \notin \{i, j\}$ 
      do
        if  $x_{ij}^* > x_{ik}^* + x_{kj}^*$  then
           $V \leftarrow V \cup \{x_{ij} \leq x_{ik} + x_{kj}\}$ 
        end if
      end for
     $R \leftarrow R \cup \{\text{random subset of } V \text{ of size } \leq p\}$ 
  until  $V = \emptyset$ 
end procedure

```

Algorithm 3: Deleting constraints when they become unnecessary

### 3.3 Deleting unnecessary constraints

Intuitively, it seems obvious that we should delete constraints from the problem when they become loose. We can also delete constraints which are tight, as long as they also happen to be basic (loose constraints are always basic). Basic constraints can be removed without affecting the current solution, which is defined as the intersection of the current set of non-basic constraints.

When constraints are deleted, stability also has to be considered. It is possible that we remove a set of constraints, re-optimize, and then find the same constraints are violated again and have to be replaced *ad infinitum*. On the other hand, if  $S$  increases with each call to the solver as it should, then there is no possibility of repeating a solution. If  $S$  fails to increase then we skip the deletion, allowing the constraint-set to build up until some progress is finally made, suggested by Padberg (1999, page 115, step 3).

Considering the dual gives us an alternative viewpoint on this problem. Re-optimizing after adding violated constraints necessarily changes the primal solution, but if  $S$  has not increased, then the dual solution has not changed and is *dually-degenerate*, i.e. at the intersection of more than the expected number of constraint boundaries, somewhat like the primally-degenerate vertices  $B, C, D, E$  of Figure 3. If we withhold information on the dual variables involved in the degeneracy, the solver's anti-degeneracy measures fail, and cycling can readily be observed.

Algorithm 3 shows the final solving routine incorporating these improvements.

## 4 Approximately rounding the LP

The solving algorithms return an optimal real solution  $\mathbf{x}^*$  to the LP (3), but as we have relaxed the integrality constraints, we have to consider that the optimal solution  $\mathbf{x}^*$  might not solve the original BIP, i.e. we may have some  $0 < x_{ij}^* < 1$ .

Interestingly, for the MUSHROOMS data set used in our experiments, we never observed any solutions (even intermediate ones) which were not integer. On the other hand, for the ADULT data set, rounding was

required. In our observation, nearly all data sets at the UCI repository are of this latter type.

Solving general Integer Programs to optimality is enormously costly. This leads us to consider approximation schemes for performing the rounding. In any case, on large data sets, we use unsampling (Section 5), which means there is little point in performing an exact rounding, since the eventual result will be an approximation anyway. We believe that better results on large data sets are obtained by spending the effort on increasing the sample size.

As there is considerable redundancy in the LP, it is natural to suggest approximating the rounding by looking at only some of the variables  $x^*$ . We use the simple probabilistic greedy approximation algorithm CCLP-PIVOT of Ailon et al. (2005), which examines about  $st/2$  of the  $x^*$ , where  $t$  is the number of clusters produced, and interestingly,  $x_{ij}^*$  is treated as the *probability* that items  $i, j$  should be clustered together, so each run may produce a different clustering. The choice of which variables to examine (*pivot selection*) also influences the result.

CCLP-PIVOT proceeds by selecting an unclustered item  $i$  at random to use as the pivot and removing it from consideration, then each unclustered item  $j$  either remains in consideration with probability  $x_{ij}^*$  or is removed and clustered with the pivot with probability  $1 - x_{ij}^*$ . See Algorithm 4, which returns  $\mathbf{C} = [C_0, C_1, \dots, C_{t-1}]$  made up of  $t$  clusters which partition items  $\{0, 1, \dots, s-1\}$ . We assume that items are randomly permuted beforehand, making it unnecessary to randomize the pivot selection.

```

procedure CCLP-Pivot
   $t \leftarrow 0$ 
   $U \leftarrow \{0, 1, \dots, s-1\}$ 
  while  $U \neq \emptyset$  do
     $i \leftarrow \min(U)$ 
     $C_t \leftarrow \{i\}$ 
     $U \leftarrow U \setminus \{i\}$ 
    for each  $j \in U$  do
      let  $r \in [0, 1)$  uniformly at random
      if  $x_{ij}^* \leq r$  then
         $C_t \leftarrow C_t \cup \{j\}$ 
         $U \leftarrow U \setminus \{j\}$ 
      end if
    end for
     $t \leftarrow t + 1$ 
  end while
end procedure

```

Algorithm 4: Assigning items to clusters with CCLP-PIVOT

## 5 Sampling and Unsampling

Although the improvements to the linear program of Section 3 substantially increase the size of problems we can solve, they do not allow us to reach the size of typical data sets for clustering problems. In order to tackle the full problem the usual approach is to take a random *sample* of points from the data set, optimize the LP corresponding to the sample, round the solution, and then *unsample*, that is, extend the consensus clustering to the remaining items.

We unsample using LOCALSEARCH, which is a simple greedy approximation algorithm proposed by Gionis et al. (2007). Items  $s, s+1, \dots, s'-1$  are considered in turn, where  $s'$  is the size of the full data set, and a locally-optimal choice is made as to which

cluster to place the item in. In our version of the algorithm,

- we evaluate the *cost*, that is, the change in  $S$  as defined in Equation (1), of placing the item in each candidate cluster;
- we update the clusters as each item is placed, before costing subsequent items;
- if placing the new item in a cluster by itself would minimize the cost, we do so.

We assume for simplicity that the  $s'$  original points are first randomly permuted and then the first  $s$  are considered the sample. We use the previous algorithms to create a consensus clustering  $\mathbf{C} = [C_0, C_1, \dots, C_{t-1}]$  for these  $s$  points. Referring to Equation (2), the cost of placing a new item  $u$  into one of these clusters, say  $C_a$ , is given by

$$\begin{aligned}
 \Delta_S(u, a) &= \sum_{v \in C_a} w_{uv}^- + \sum_{v \notin C_a} w_{uv}^+ \\
 &= \sum_{v \in C_a} (w_{uv}^- - w_{uv}^+) + \sum_v w_{uv}^+ \\
 &= \sum_{v \in C_a} (1 - 2w_{uv}^+) + \text{constant} \\
 &= |C_a| - 2 \sum_{v \in C_a} w_{uv}^+ + \text{constant}. \quad (4)
 \end{aligned}$$

We drop the constant, as it's irrelevant to choosing the best  $a$ , and then examine the contribution from each input clustering  $\pi_0, \pi_1, \dots, \pi_{m-1}$ . Let  $\mathbf{D}^k = [D_0^k, D_1^k, \dots, D_{t_k-1}^k]$  be the clusters of  $\pi_k$ , with adjacency  $y_{uv}^k$ , and consider a function  $d$  describing per input clustering which cluster number  $u$  falls into, i.e. let  $u \in D_{d(k)}^k$ . Then

$$\begin{aligned}
 \Delta_S(u, a) &= |C_a| - 2 \sum_{k=0}^{m-1} \sum_{v \in C_a} (1 - y_{uv}^k) \\
 &= |C_a| - 2 \sum_{k=0}^{m-1} |C_a \cap D_{d(k)}^k|. \quad (5)
 \end{aligned}$$

Supposing  $C_a = \emptyset$ , obviously  $\Delta_S(u, a) = 0$ , so we create a new cluster if placing in the best existing cluster is more costly than this. The entire procedure is summarized as Algorithm 5.

```

procedure Unsample
  for  $i = s$  to  $s' - 1$  do
     $a \leftarrow \operatorname{argmin}\{\Delta_S(u, a) : a = 0, 1, \dots, t-1\}$ 
    if  $\Delta_S(u, a) > 0$  then
       $C_t \leftarrow \{i\}$ 
       $t \leftarrow t + 1$ 
    else
       $C_a \leftarrow C_a \cup \{i\}$ 
    end if
  end for
end procedure

```

Algorithm 5: Extending the clustering on  $s$  items to all  $s'$  items

In the implementation we store the  $|C_a \cap D_b^k|$  as a series of sparse matrices, and maintain them as items are added to clusters. Whilst the evaluation of Equation 5 costs us  $O(\log(t_k))$  time to access each cell of the sparse matrix, it's a reasonable compromise between using a dense matrix and the alternative of implementing Equation 4 directly (which would be appropriate for very large/small clusters respectively).

	POSTOPERATIVE	LYMPHOGRAPHY	MUSHROOMS	ADULT
items	90	148	8124	48842
clusterings	8	19	22	14
clusters	9	7 or 8	10*	330*
cluster size	10*	20*	812*	1481*
type	categorical	categorical	categorical	heterogeneous
missing values	yes	no	yes	no
each item is a	post-operative patient	oncology patient	mushroom species	adult U.S. citizen

Table 1: Characteristics of each data set, \* approximate value

data set	sample	ALLCONSTRAINTS		ALLVIOLATED		SOMEVIOLATED		DELETEDLOOSE	
		time	memory	time	memory	time	memory	time	memory
MUSHROOMS (8124 items)	256	112.5s	2752.3Mb	5.2s	298.3Mb	0.7s	57.6Mb	0.6s	37.2Mb
	512	–	> 4Gb	141.3s	2097.0Mb	6.9s	235.9Mb	6.3s	134.2Mb
	1024	–	> 4Gb	–	> 4Gb	76.6s	857.2Mb	69.5s	497.4Mb
	2048	–	> 4Gb	–	> 4Gb	> 3hrs	–	1145.0s	1869.3Mb
ADULT (48842 items)	32	0.0s	36.0Mb	0.0s	28.3Mb	0.0s	28.3Mb	0.0s	28.2Mb
	64	0.7s	99.1Mb	0.2s	32.8Mb	0.1s	31.2Mb	0.1s	30.1Mb
	128	57.6s	630.6Mb	25.6s	73.4Mb	22.0s	47.9Mb	8.6s	37.1Mb
	256	> 3hrs	–	> 3hrs	–	> 3hrs	–	2160.0s	67.3Mb

Table 2: Time and space improvement per algorithm and data set

## 6 Experiments

### 6.1 Data sets

We tried our algorithms on several data sets available from the UCI Machine Learning Repository (Asuncion & Newman 2007), some features of which are summarized in Table 1.

The MUSHROOMS data set, which we also used in our previous research (Bertolacci & Wirth 2007), has pure categorical data, e.g. there is a column *cap shape* having values *bell*, *conical*, and so on. We use each column  $k$  to define an input clustering  $\pi_k$  such that  $y_{uv}^k = 0$  if the values in column  $k$  for rows  $u$  and  $v$  are identical. The smaller POSTOPERATIVE and LYMPHOGRAPHY data sets are also pure categorical data. The indicated data sets have a few missing values, denoted ‘?’, which for simplicity we treat as a legitimate data value.

The ADULT data set has heterogeneous data. Most columns such as *sex* are treated identically to MUSHROOMS, but some columns have continuous values which should be clustered by an auxiliary clustering algorithm before participating in the consensus clustering. In our case the auxiliary algorithm is simplistic, we merely quantize the *age* to the nearest 5 years, and monetary values to the nearest \$5,000, and then treat them as categorical data.

The *class* column (e.g. for MUSHROOMS, it is the *edibility* column) is not used, because the data sets are intended for machine learning experiments, i.e. predicting this value from the others, whereas we are only interested in the raw characteristics of each item.

### 6.2 Results

We used iLOG CPLEX 10 as the LP solver for the experiments. Table 2 shows the time and space required for applying the solving algorithms described previously to different samples of the MUSHROOMS and ADULT data sets. This experiment was run on a 1.9 GHz AMD Phenom II dual-core with 4 Gbytes of RAM. We tried each algorithm and data set combination with increasing problem sizes. This illustrates how the time and space requirements are growing with problem size, and the exact improvement due to each improved version of the algorithm. Unsampling was not performed in this experiment.

Missing values in Table 2 illustrate the increase in

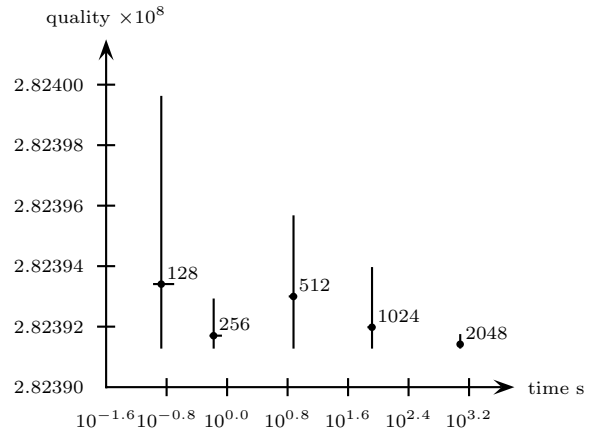


Figure 4: Changing the sample size with MUSHROOMS

the problem sizes that can be tackled with our more advanced algorithms. We separate cases that ran out of memory (4Gb) or time (3hrs). Previously, with the basic ALLCONSTRAINTS algorithm, it was not possible to solve MUSHROOMS instances larger than 256 items due to lack of memory. We can now easily solve (sub-)problems of up to 2,048 items, and the results are optimal on every sample that we’ve seen so far.

The ADULT problem is much more difficult to solve than MUSHROOMS, though memory requirements are similar (compare the performance of each on a 256-item sample). The principal difference appears to be the larger number of clusters produced, which may make the LPs more difficult to solve. Indeed, none of the LP solutions of the ADULT samples give integer solutions, as opposed to MUSHROOMS. Still, we can now easily solve ADULT sub-problems twice as large as was previously possible, and larger problems could also conceivably be tackled given enough time.

We then investigated the quality of the resulting clusterings. Figures 4 and 5 show the results of Algorithm 3 (DELETEDLOOSE) on increasing sample sizes for each data set. The LP solutions were rounded with CCLP-PIVOT before being unsampled to the full data set, and the resulting clustering quality ( $S$ ) and execution time reported.

The data points shown in the graphs are the result



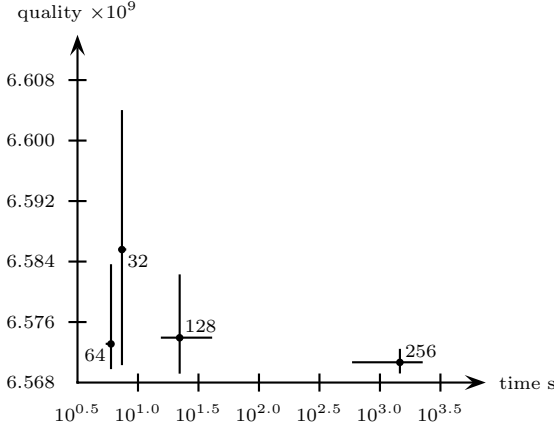


Figure 5: Changing the sample size with ADULT

of averaging 5 different runs, each based on a different permutation of the data items (this affects the rounding, the unsampling, and possibly the LP solution). Error bars show the minimum and maximum times/costs encountered in the 5 runs.

Several of the data points appear to be anomalous. Without devoting unreasonable amounts of computer time to increasing the number of sample runs, we cannot tell whether this is a random effect or whether the sample sizes are interacting with the data sets, algorithms or hardware in some way. As the trends seem evident we didn't pay any attention to this issue.

The most interesting feature of these results is that they are asymptotic, with the clustering quality approaching what seems to be a lower limit, despite the relatively small sample sizes. This suggests that the CONSENSUS CLUSTERING problem might not be as hard as previously thought, at least on typical data sets. For example, a particular clustering that is often produced for the MUSHROOMS data set has cost 282,391,274, and we believe this may be (one of) the optimal clustering(s).

Whilst good quality clusterings can sometimes be found without much effort, inspecting the error bars it is clear that our advanced approach to tackling the LP is necessary for *reliably* producing good quality output. On the other hand, if the resources aren't available to solve the larger sample sizes that are now possible with our algorithms, then referring to Table 2, we can solve any sample faster and in less space than before, yielding a much better approximation in the time and space available.

## 7 Exactly rounding the LP

While modern MIP solvers are very powerful, including many specialized techniques to tackle BIP problems, since the problems (3) in their raw form are too big to solve as an LP, we cannot hope to use a MIP solver directly on any but the smallest problem sizes. Therefore, we tried a *branch-and-bound* approach utilizing our Algorithm 3 to solve the branch-and-bound subproblems, warm-starting each time with the solution and constraint-set from the previous call.

We make use of the fast approximate rounding Algorithm 4 (CCLP-PIVOT) as a diving algorithm to generate good candidate integer solutions. This is motivated by our conjecture of Subsection 6 that optimal solutions may quite frequently be produced by the approximation algorithms.

Initially we tried performing  $(s-1)s/2$  CCLP-PIVOT attempts per iteration (i.e. the number of variables), which produced good results. Branching

depths of up to 4 were observed. Interestingly, in some cases the optimal solution was never found during the test-roundings, but the approximate solution proved good enough to guide the branching and find the optimal solution by variable-fixing.

Considering the importance of good bounding, we then tried applying as many passes of LOCALSEARCH as necessary to polish the solutions. This is essentially Algorithm 5, but un-clustering and re-clustering each item in turn, rather than considering new items only. We then need to perform only  $s$  rounding attempts (i.e. the number of items) per iteration, which is in fact quite generous, as typically the polishing is so good that the optimal solution is found after  $< 10$  attempts, whereupon the task is simply to prove its optimality, which may require little branching.

The entire procedure with polishing is shown as Algorithm 6, essentially it is iterative deepening branch-and-bound with strong branching to reduce the search space, i.e. the unlikely branch is tried first, and then the likely branch is tried. The  $F_d$  refers to the forcing constraints at level  $d$ , note that  $F_{d+1}$  inherits from  $F_d$  but not vice versa. Also, each call to Algorithm 3 passes in the value of  $S^B$  and the process of adding constraints and re-solving is aborted when  $S(\mathbf{x}^d)$  exceeds this, since it can never decrease (we omit this detail from Algorithm 6 for simplicity).

```

function BranchAndBound( $d$ )
   $\mathbf{x}^d \leftarrow \operatorname{argmin}\{S(\mathbf{x}) : \mathbf{x} \text{ satisfies } F_d \cup R\}$ 
    by DELETELOOSE warm-started with
     $\mathbf{x}, R$  from the previous invocation
  if  $F_d \cup R$  infeasible or  $S(\mathbf{x}^d) > S^B$  then
    return false
  end if
  if  $d < d_{\max}$  then
    for each  $0 \leq i < j < s$  with  $0 < x_{ij}^d < 1$  do
       $F_{d+1} \leftarrow F_d \cup \{x_{ij} = 1 - x_{ij}^B\}$ 
      if BranchAndBound( $d+1$ ) = false then
         $F_d \leftarrow F_d \cup \{x_{ij} = x_{ij}^B\}$ 
         $\mathbf{x}^d \leftarrow \operatorname{argmin}\{S(\mathbf{x}) : \mathbf{x} \text{ satisfies } F_d \cup R\}$ 
          by DELETELOOSE warm-started with
           $\mathbf{x}, R$  from the previous invocation
        if  $F_d \cup R$  infeasible or  $S(\mathbf{x}^d) > S^B$  then
          return false
        end if
      end if
    end for
  end if
  return true
end function

function ExactRounding
   $F_1 \leftarrow \emptyset$ 
   $R \leftarrow \emptyset$ 
   $S^B \leftarrow +\infty$ 
  for  $d_{\max} = 1$  to  $\infty$  do
    BranchAndBound(1)
    if  $x_{ij}^1 \in \{0, 1\} \forall 0 \leq i < j < s$  then
      return  $\mathbf{x}^1$ 
    end if
     $(\mathbf{x}^B, S^B) \leftarrow \text{best of } (\mathbf{x}^B, S^B) \text{ and } s \text{ roundings}$ 
      of  $\mathbf{x}^1$  by CCLP-PIVOT + LOCALSEARCH
    end for
end function

```

Algorithm 6: Performing exact rounding by iterative-deepening branch-and-bound

At depth  $d$  we solve the LP to obtain  $\mathbf{x}^d$  and prune the search if this produces a (real) result  $S(\mathbf{x}^d)$  worse than the best known integer solution  $S^B$ . We then

data set	sample	quality	clusters	CPLEX MIP		clusters	EXACTROUNDING	
				time	memory		time	memory
POSTOPERATIVE	90	13059	9	1395.0s	175.9Mb	9	56.2s	15.4Mb
LYMPHOGRAPHY	148	83967	—	> 3hrs	—	7	134.6s	23.1Mb
ADULT (48842 items)	32	2664	11	0.1s	8.1Mb	11	0.0s	4.4Mb
	64	11260	14	3.3s	35.0Mb	14	0.1s	5.9Mb
	128	44788	—	> 3hrs	—	25	59.1s	18.2Mb
	256	181410	—	> 3hrs	—	31	1124.0s	44.7Mb

Table 3: Exact rounding time and space improvement over CPLEX MIP

consider each fractional  $x_{ij}^d$  in turn, and recurse to see if forcing  $x_{ij}^d$  to the opposite value  $1 - x_{ij}^B$  to the best current solution will make the (real) solution worse than the bound,  $S^B$ . If so, we can fix  $x_{ij}^d$  to the value  $x_{ij}^B$  in the current best solution and update the  $\mathbf{x}^d$ , before proceeding to check the remaining  $x_{ij}^d$ .

### 7.1 Exact-rounding results

Results from Algorithm 6 are promising. We tried it on the smaller data sets POSTOPERATIVE and LYMPHOGRAPHY for which taking a sample was not required. We also tried it on increasing samples drawn from the ADULT data sets. Referring to Table 3, CPLEX’s inbuilt MIP solver could only solve problems up to 90 items. We can solve these in greatly reduced time and space, and we can comfortably solve ADULT (sub-)problems up to 256 items.

This table was generated with mostly default settings for CPLEX’s MIP solver. We turned on aggressive Gomory Fractional Cut-generation and turned off all other cuts. This was a result of earlier tuning experiments, although in generating this table we did not see it apply cuts. We generated all constraints initially but we placed them in a lazy constraint-pool, which seems similar to our Algorithm 1 (ALLVIOLATED), only without the memory savings or the improvements of SOMEVIOLATED or DELETEDLOOSE.

CPLEX’s handling of lazy constraints was adequate for the small problems we consider here. The real problem with CPLEX MIP seems to be its inability to generate good candidate solutions. In many cases no candidate integer solutions had been generated at all after several hours, hence the advantage of the problem-specific rounding and polishing as in our advanced Algorithm 6 (EXACTROUNDING).

## 8 Conclusions and further work

The best current method we are aware of for solving CONSENSUS CLUSTERING problems, illustrated by the experiments of Bertolacci & Wirth (2007), is the combination of solving the linear relaxation (3) of a Boolean Integer Program, with approximate rounding, sampling, and unsampling.

By the careful formulation of a suitable LP for use in a Dual Simplex LP solver, and the use of advanced Linear Programming techniques to reduce the size of the problem tackled by the solver, we are able to solve much bigger CONSENSUS CLUSTERING problems to optimality than was previously possible.

We can consistently solve (sub-)problems of size up to 2,048 sampled from the MUSHROOM data set to optimality. Interestingly, for the ADULT data set the largest sub-problem whose LP we can tackle in a reasonable time is only 256, illustrating the very different nature of the data sets. For building consensus clusterings on large data sets, our methods allow us to solve the same sample size much faster, and/or

improve the resulting consensus clustering by using a larger sample.

By using approximate rounding with the LP formulation we can speedily generate very good consensus clusterings. We use this as a diving heuristic to create a strong branch-and-bound approach to finding optimal consensus clusterings. We generate for the first time an optimal consensus clustering for the complete (albeit small) data sets POSTOPERATIVE and LYMPHOGRAPHY.

We plan to extend our research to more sophisticated LP solution methods, in particular a more sophisticated policy for adding the triangle inequality constraints rather than the random selection that we currently use. We also plan to investigate more sophisticated *branch-and-cut* approaches to generating optimal solutions to consensus clustering problems.

### Acknowledgments

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council.

### References

- Ailon, N., Charikar, M. & Newman, A. (2005), Aggregating inconsistent information: ranking and clustering, in ‘STOC ’05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing’, ACM, New York, NY, USA, pp. 684–693.
- Asuncion, A. & Newman, D. (2007), ‘UCI machine learning repository’.  
**URL:** <http://www.ics.uci.edu/~mllearn/MLRepository.html>
- Bansal, N., Blum, A. & Chawla, S. (2002), Correlation clustering, in ‘Machine Learning’, pp. 238–247.
- Bertolacci, M. & Wirth, A. (2007), Are approximation algorithms for consensus clustering worthwhile?, in ‘SDM’, SIAM.
- Charikar, M., Guruswami, V. & Wirth, A. (2003), ‘Clustering with qualitative information’, *Foundations of Computer Science, Annual IEEE Symposium on* **0**, 524.
- Cohen, W. & Richman, J. (2001), Learning to match and cluster entity names, in ‘In ACM SIGIR-2001 Workshop on Mathematical/Formal Methods in Information Retrieval’.
- Cohen, W. W. & Richman, J. (2002), Learning to match and cluster large high-dimensional data sets for data integration, in ‘KDD ’02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining’, ACM, New York, NY, USA, pp. 475–480.

- Coleman, T., Saunderson, J. & Wirth, A. (2008), Spectral clustering with inconsistent advice, *in* W. W. Cohen, A. McCallum & S. T. Roweis, eds, 'ICML', Vol. 307 of *ACM International Conference Proceeding Series*, ACM, pp. 152–159.
- Emanuel, D. & Fiat, A. (2003), Correlation clustering minimizing disagreements on arbitrary weighted graphs, *in* 'Proceedings of the 11th Annual European Symposium on Algorithms', Springer, pp. 208–220.
- Filkov, V. & Skiena, S. (2003), Integrating microarray data by consensus clustering, *in* 'ICTAI '03: Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence', IEEE Computer Society, Washington, DC, USA, p. 418.
- Gionis, A., Mannila, H. & Tsaparas, P. (2007), 'Clustering aggregation', *ACM Trans. Knowl. Discov. Data* **1**(1), 4.
- Mirkin, B. G. & Chernyi, L. B. (1970), 'On Measurement of Distance Between Partitions of a Finite Set of Units', *Automation and Remote Control* **31**, 786–792.
- Padberg, M. (1999), *Linear Optimization and Extensions*, Vol. 12 of *Algorithms and Combinatorics*, 2 edn, Springer Verlag.
- Rand, W. M. (1971), 'Objective criteria for the evaluation of clustering methods', *Journal of the American Statistical Association* **66**(336), 846–850.  
**URL:** <http://www.jstor.org/stable/2284239>
- Wakabayashi, Y. (1998), 'The Complexity of Computing Medians of Relations', *Resenhas* **3**(3), 323–349.





# Estimating Set Intersection using Small Samples \*

Henning Köhler

School of Information Technology and Electrical Engineering,  
The University of Queensland,  
Brisbane, Australia  
Email: [henning@itee.uq.edu.au](mailto:henning@itee.uq.edu.au)

## Abstract

The similarity of two sets  $A, B$  can be measured by the size of their intersection  $A \cap B$ , relative to the size of  $A$  and  $B$ . The classic measure here is resemblance or Jaccard similarity, but other useful measures (e.g. subset containment) can be derived from intersection size as well. For large and/or many sets, exact computation of intersection size can be expensive though, and requires transmitting entire sets if they are distributed. For this reason a number of different sampling techniques have been developed, which allow us to estimate intersection size (and derived measures) efficiently from the intersection size of smaller sample sets. However, while existing estimation formulas are intuitive and unbiased, they can be quite inaccurate when samples are small. We show that by using more advanced estimation techniques, we can significantly reduce sample sizes without compromising accuracy, or conversely, obtain more accurate results from the same samples.

## 1 Introduction

The issue of measuring the similarity of sets arises in many different application areas. Most notably, it has received attention in the areas of information retrieval (Manber (1994), Broder (1997), Broder et al. (1997), Broder (2000)), and data integration (Dasu et al. (2002), Rahm & Bernstein (2001)). In the first case, text documents are replaced by sets of substrings (“chunks”) occurring within them (e.g. words, q-grams, v-grams, ...), and the similarity of documents is then measured by the similarity of the resulting sets (i.e., by how many substrings they share). Among many other applications, this allows plagiarism detection, lineage tracing, and pruning of near-duplicate search results. For data integration, set similarity is used by value-based matchers, which identify related attributes (e.g. via foreign key) by the similarity of their value sets.

Closely related is the area of join size estimation in databases, which has received much attention in the past, e.g. Olken & Rotem (1986), Ganguly et al. (1996), Chaudhuri et al. (1999), Haas & Koenig (2004) among many others. Instead of estimating in-

tersection size of two sets, the goal is to estimate

$$\sum_{x \in A \cap B} mult_A(x) \cdot mult_B(x)$$

for two multi-sets  $A, B$  (here  $mult_A(x)$  denotes the multiplicity of  $x$  in  $A$ ).

Given two sets  $A, B$ , determining  $|A \cap B|$ , the size of their intersection, is not hard. Once obtained, we can then derive their resemblance

$$res(A, B) := \frac{|A \cap B|}{|A \cup B|}$$

subset containment

$$sub(A, B) := \frac{|A \cap B|}{|A|}$$

or other similarity measures. However, the problem becomes more difficult since we are typically considering not just two set, but (depending on the application) thousands or even millions of them, and sets can be huge. Furthermore, we are often dealing with sets which are distributed over multiple servers, and comparing sets for similar elements typically requires us to transport all data involved to a central location for analysis. To reduce computation costs and avoid shipping the entire data set, one can use sampling techniques, which generate a small signature for each attribute. In the distributed case, these signatures can be generated locally and then shipped to a central location for comparison.

On the downside, we only obtain the intersection size of sample sets. From this we must then estimate the intersection size for the actual value sets, which necessarily introduces errors. While several excellent sampling techniques have been developed, all of them rely on rather basic techniques for estimating the intersection size of the original sets from the intersection size of their samples. In this paper we propose to use of a new probability based method for this task, and show that this allows us to reduce the sample size by up to factor four (for uniform distributions, or by even higher factors for non-uniform ones) while achieving the same accuracy. Alternatively we obtain much more accurate results from the same sample sets. Our method requiring no changes to the sampling algorithms, and thus should be easy to implement with existing systems.

The rest of the paper is organized as follows. In Section 2 we introduce and compare sampling techniques. For these we then develop different functions for estimating intersection size of the original sets from samples in Section 3, with focus on uniform distributions. Non-uniform distributions of intersection sizes are discussed in Section 4. We then compare the estimation functions developed with existing ones in Section 5, and conclude in Section 6.

\* This work is partly supported by CSIRO’s Water for a Healthy Country Flagship.

Copyright ©2010, Australian Computer Society, Inc. This paper appeared at the Thirty-Third Australasian Computer Science Conference (ACSC2010), Brisbane, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 102, B. Mans and M. Reynolds, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

## 2 Sample Generation

To obtain accurate results with small samples, the sampling approach needs to be synchronized. Independent random sampling would mean that even identical value sets may produce different, possibly even disjoint samples. To make sampling practical, it must be possible to

1. preserve relative overlap
2. take a sample without knowing anything about other sets
3. only sample each set once

We will discuss three different sampling techniques (plus a minor variation) for this. A central observation is that all of them can be modeled as randomly selecting elements from the union of the two sets that we wish to compare, and thus our method for estimating intersection size (which is based on this model) can be directly applied to all of them. While we only describe sample generation for two sets, the extension to multiple sets is straight-forward: Each set is sampled once, and whenever we wish to compare two sets we compare their respective samples. Note that in the literature the terms ‘sample’ and ‘signature’ are often used interchangeably (and we will do so as well), although strictly speaking the signature may contain other data (e.g. the size of the original set) as well.

Broder (1997) describes two approaches for creating efficient signatures. The first uses a (pseudo-)random total ordering on the set of all values, and takes as signature the  $s$  smallest values in each set, for some constant  $s$ . It is important to note that the same ordering is used for all sets. Then the signatures  $S(A), S(B)$  of two sets  $A, B$  can be compared as follows: For  $u := \min(\max(S(A)), \max(S(B)))$  we have

$$S(A) \cap S(B) = \{x \in A \cap B \mid x \leq u\}$$

Thus any value above  $u$  in  $S(A), S(B)$  is ‘useless’ for determining intersection sizes, so the *effective* signatures used for comparing  $A$  and  $B$  are

$$\begin{aligned} S'(A) &:= \{x \in S(A) \mid x \leq u\} \\ S'(B) &:= \{x \in S(B) \mid x \leq u\} \end{aligned}$$

We can then interpret results about the intersection size  $|S'(A) \cap S'(B)| = |S(A) \cap S(B)|$  as randomly selecting  $|S'(A) \cup S'(B)|$  distinct values from  $A \cup B$ , and getting  $|S'(A) \cap S'(B)|$  values in  $A \cap B$ . Note that although the signature size is fixed by  $s$ , the effective signature size varies.

The second approach in Broder (1997), originally proposed in Manber (1994), takes as signature all values whose hash value is divisible by a constant  $m$ . Here we have

$$S(A) \cap S(B) = \{x \in A \cap B \mid x = 0 \pmod m\}$$

We can again see this as randomly selecting  $|S(A) \cup S(B)|$  distinct values from  $A \cup B$ . However, the approach becomes troublesome when sets vary greatly in size, which is often the case. Then a small constant  $m$  generates large signatures for large sets and thus high transport and computation costs, while a large constant generates small signatures for small sets, making estimates for their intersection size inaccurate.

To address this problem, Broder proposes a variant of the second approach, where the value  $m = 2^i$  for some  $i$  depends on the size of the value set, chosen

to generate signatures of roughly equal size. This allows for more accurate comparison of small sets without causing signatures of large sets to grow too large. However, when comparing two signatures which use different constants  $i_A, i_B$ , then only the values which are  $0 \pmod{2^{\max(i_A, i_B)}}$  can contribute to the intersection, so we have again ‘useless’ values, and effective signatures

$$\begin{aligned} S'(A) &:= \{x \in S(A) \mid x = 0 \pmod{2^{i_B}}\} \\ S'(B) &:= \{x \in S(B) \mid x = 0 \pmod{2^{i_A}}\} \end{aligned}$$

The methods of signature generation discussed to far can all be performed quickly in near-linear time, and comparing two signatures with respect to intersection size amounts to randomly selecting  $|S(A) \cup S(B)|$  or  $|S'(A) \cup S'(B)|$  distinct values from  $A \cup B$ . In Dasu et al. (2002) a different approach is used known as ‘min hashing’, originally proposed by Broder in Broder (2000) for finding near-duplicate documents. Instead of taking as signature the  $s$  smallest elements w.r.t. a single random ordering, their signature consists of the smallest values w.r.t.  $s$  different random orderings (typically implemented as hash functions). Then given the signatures for two sets  $A, B$ , the minimal value  $\min_\sigma(A \cup B)$  in  $A \cup B$  w.r.t. an ordering  $\sigma$  can be computed as  $\min_\sigma(\min_\sigma(A), \min_\sigma(B))$ . This value lies in  $A \cap B$  iff  $\min_\sigma(A) = \min_\sigma(B)$ . Thus we are effectively selecting a single value from  $A \cup B$  at random for each of the  $s$  random orderings. A nice property of this sampling approach is that one can easily group minimal values into blocks, and then compare signatures block-wise by reducing each block to a single hash value Broder (2000). This is a trade-off between speed and accuracy, which is very favorable for finding pairs with high resemblance threshold (near-duplicates).

Signature size is constant for min hashing, and at the same time we have no ‘useless’ elements. However, of every pair of elements  $\min_\sigma(A), \min_\sigma(B)$  we are effectively using only one, which gives us an effective sample size equal to one of the signatures, whereas the approaches in Broder (1997) provide effective sample sizes at least as large as one of the signatures, and usually larger. Furthermore, multiple independent sampling has a slightly larger variance than selecting the full sample set at once (without chance of duplicates). Finally, computation of the signature can take up to  $s$  times longer than the approaches proposed in Broder (1997), as we need to compare elements w.r.t.  $s$  different, independent orderings (note that this is only an upper bound, and that the actual difference is typically much smaller).

Lastly, the authors of Dasu et al. (2002) also describe measures for set similarity based on string similarity, when set elements are strings. This is done by substituting each string value by the  $q$ -grams appearing in it, so the problem to be solved is exactly the same as before, i.e., we need to estimate intersection size for sets.

For join size estimation, sampling techniques are very different. While the above techniques could in principal be used as well (by storing the multiplicity of sampled elements, as suggested by Dasu et al. (2002)), this can lead to very inaccurate estimates as frequent elements are easily missed. In this paper we will focus exclusively on the set intersection problem, and consider join size estimation a topic for future work.

## 3 Estimating Set Similarity

Computing the pairwise intersection size of multiple sample sets can be done efficiently using appropriate

index structures (Bauckmann et al. (2007), Broder (2000)). However, the exact methods for doing this are not relevant to our approach, so we won't discuss them. Instead we will just assume that the intersection size of the (effective) signatures of two sets  $A, B$ , denoted by  $S(A), S(B)$ , is known, and we now wish to estimate the size of the intersection  $A \cap B$ .

For simplicity we shall model the signature generation process as randomly (uniformly and independent) selecting  $|S(A) \cup S(B)|$  values from  $A \cup B$ . There are two small issues with this, which we will discuss in the following - however, neither of them has any significant impact on the accuracy of our method (and the second issue is in fact just a special case of the first). For a start, we pretend that  $|S(A) \cup S(B)|$  is fixed, although it usually isn't. In practice, it depends on the different sampling parameters (which are sometimes chosen dynamically depending on the set sampled, e.g. the parameter  $i$  in "mod  $2^i$ " sampling) as well as the actual sets  $A, B$ . Second, the min hashing method is more accurately modeled as selecting a single element from  $A \cup B$   $s$  times. In contrast, we discard "duplicate" results (i.e. where the minimal element in  $A \cup B$  is the same for multiple random orderings) and pretend they were never selected. This actually has no direct impact on our estimate (since all values are equally likely to be duplicated), but causes the signature size after duplicate removal to vary, even though  $s$  is constant.

We choose to use this slightly inexact model since it allows us to treat all sampling methods uniformly, and independently of specific implementation issues (e.g. how sampling parameters are chosen). Also it simplifies the analysis, without any noticeable decrease in accuracy. It is worth noting though that our approach can equally well be applied to more exact, specialized models, although the resulting analysis may be less pretty.

To simplify notation, we will use the following abbreviations:

$$\begin{aligned} a &:= |S(A) \setminus S(B)| \\ b &:= |S(B) \setminus S(A)| \\ x &:= |S(A) \cap S(B)| \end{aligned}$$

and let  $S(A), S(B)$  denote the *effective* signatures, as discussed in Section 2.

### 3.1 Minimal Standard Error

Whenever we are estimating the intersection size based on samples, we expect to make errors. In order to compare different estimation functions, we use the notion of *standard error*, similar to the standard deviation of random variables.

**Definition 1.** Given a random variable  $\phi$  estimating  $X$ , the standard error is

$$\sqrt{E((\phi - X)^2)}$$

where  $E(\dots)$  denotes the expected value of the argument.

Here an estimation function  $\phi$  turns into a random variable as we give it constants as well as random variables as input: The size of the sets  $A, B$  are constants, while the size of their signatures  $S(A), S(B)$  and the intersection  $S(A) \cap S(B)$  varies. For simplicity we assume that the size of  $S(A) \cup S(B)$  is fixed, so that we are always randomly selecting a fixed number of values from  $A \cup B$ . Note that for unbiased estimation functions  $\phi$ , i.e., where  $E(\phi) = X$ , the standard error of  $\phi$  becomes exactly the standard deviation.

The probability of getting a certain sample pair depends on the value for  $X = |A \cap B|$ . If we fix  $X = |A \cap B|$ , then there exists a perfect estimation function: The constant function returning  $X$  regardless of input. Since we don't know what  $X$  is in the first place though, trying to optimize estimation functions for a fixed value of  $X$  is pointless. Instead we will assume (for now) that all possible values for  $X$  are equally likely, and try to minimize the standard error. While a minimal standard error may not always be exactly what we want, we would claim that minimizing it also provides good results for many other reasonable error functions. And should the difference in optimization objective really be unacceptable, it is always possible to perform a similar analysis for some other target function.

Now let  $p(a, b, x|X)$  denote the probability to get  $a, b, x$  elements from  $A \setminus B, B \setminus A, A \cap B$ , respectively when randomly selecting  $a+b+x$  elements from  $A \cup B$ , when  $|A \cap B| = X$ . Then we have

$$p(a, b, x | X) = \frac{\binom{|A|-X}{a} \cdot \binom{|B|-X}{b} \cdot \binom{X}{x}}{\binom{|A|+|B|-X}{a+b+x}} \quad (1)$$

Given  $|A|, |B|$  and fixing  $n := |S(A) \cup S(B)|$ , the standard error for an estimation function  $\phi$  can be calculated as the square root of

$$\sum_{X=0}^{\min(|A|, |B|)} p(X) \cdot \sum_{\substack{a+b+x=n \\ a \leq |A|-X \\ b \leq |B|-X \\ x \leq X}} p(a, b, x|X) \cdot (\phi(a, b, x) - X)^2 \quad (2)$$

Rearranging terms, this can be rewritten as

$$\sum_{\substack{a+b+x=n \\ a \leq |A| \\ b \leq |B| \\ x \leq |A|+|B|-n}} \sum_{X=x}^{\min(|A|-a, |B|-b)} p(X) \cdot p(a, b, x|X) \cdot (\phi(a, b, x) - X)^2 \quad (3)$$

It is now easy to determine an optimal approximation function  $\phi$  which minimizes (3):  $\phi(a, b, x)$  must minimize the term

$$\sum_{X=x}^{\min(|A|-a, |B|-b)} p(X) \cdot p(a, b, x|X) \cdot (\phi(a, b, x) - X)^2 \quad (4)$$

This minimum is reached for the conditional expected value of  $X$  (given  $a, b, x$ ):

$$\phi_{\min}(a, b, x) := \frac{\sum_{X=x}^{\min(|A|-a, |B|-b)} p(X) \cdot p(a, b, x|X) \cdot X}{p(a, b, x)} \quad (5)$$

where

$$p(a, b, x) = \sum_{X=x}^{\min(|A|-a, |B|-b)} p(X) \cdot p(a, b, x|X)$$

If we now assume all possible values for  $X$  to be equally likely, (5) simplifies to

$$\phi_{\min}(a, b, x) := \frac{\sum_{X=x}^{\min(|A|-a, |B|-b)} p(a, b, x|X) \cdot X}{\sum_{X=x}^{\min(|A|-a, |B|-b)} p(a, b, x|X)} \quad (6)$$

This can be seen as the best possible estimate for the value  $X$ , thus providing a lower bound for the error of

any estimation function  $\phi$ . Of cause,  $\phi_{min}$  itself could be used as an estimation function, but the price is the rather high cost for computing it. Note though that it is not necessary to evaluate the term  $p(a, b, x|X)$  appearing in (6). Instead it suffices to find the factor between consecutive values of  $p(a, b, x|X)$ , which is easier to compute:

$$\frac{p(a, b, x|X+1)}{p(a, b, x|X)} = \frac{(|A| - a - X) \cdot (|B| - b - X)}{(|A| - X) \cdot (|B| - X)} \cdot \frac{(X+1) \cdot (|A| + |B| - X)}{(X-x+1) \cdot (|A| + |B| - (a+b+x) - X)} \quad (7)$$

However, for large values of  $|A|, |B|$  we still have the problem that the number of summands in (6) is close to  $\min(|A|, |B|)$ . We can tackle this by approximating

$$\phi_{min}(|A|, |B|, a, b, x) \approx \phi_{min}\left(\frac{|A|}{c}, \frac{|B|}{c}, a, b, x\right) \cdot c \quad (8)$$

for some constant  $c$ , which reduces the number of summands in (6) at the cost of a relatively small error, at least for properly chosen  $c$ . We found a suitable choice for  $c$  to be

$$c = \frac{1}{k} \cdot \min\left(\frac{|A|}{|S(A)|}, \frac{|B|}{|S(B)|}\right)$$

with  $k \in [2, 5]$  determining the size factor between reduced set (for estimation purposes) and signature. Larger  $k$  increase accuracy, but at the price of higher computation cost.

Note that this approximation has a corresponding interpretation as a simple technique for numerical integration. Computing  $p(a, b, x)$  is effectively done by taking the (discrete) integral over  $p(a, b, x|X)$  (or over  $p(X) \cdot p(a, b, x|X)$  for non-uniform distributions). Here (8) corresponds to taking larger steps - of size  $c$  instead of 1 - in the integration process.

Another step towards reducing computation costs is to ignore small summands in (6). Consider e.g. the probability distribution shown in Figure 1. Instead of

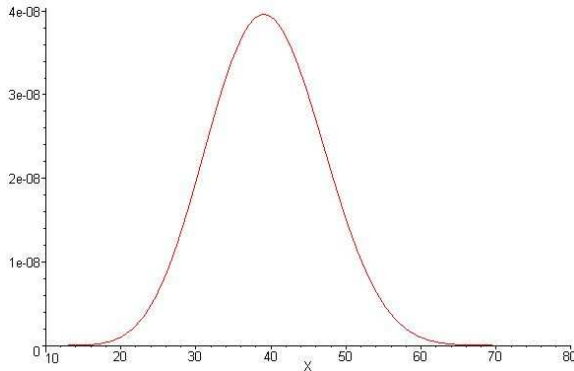


Figure 1: distribution of  $p(a, b, x|X)$  for  $|A| = 100, |B| = 200, a = 20, b = 10, x = 10$

computing summands for the entire range, it suffices to compute only those in the 20 – 60 range. The proper range can easily be found by starting at the peak around 40 (we will discuss how peaks can be found in Section 3.2), and computing summands in both directions until they fall below some threshold.

Note that it is not necessary to compute  $p(a, b, x|X)$  at the peak (let's call it  $X_{max}$ ) - we can simply use  $p(a, b, x|X_{max}) = 1$  (or some other constant other than 0). When deriving other probabilities using (7), this error is propagated, so that all summands in (5) or (6) carry the same error factor  $p(a, b, x|X_{max})$ , which cancels itself out.

### 3.2 Maximum Likelihood Estimate

Another solution that we mentioned earlier, is to take the peak of the probability distribution for  $p(a, b, x|X)$  as estimate for  $X$ . If we consider all configurations  $\{|A \cap B| = 0, 1, 2, \dots\}$  to be equally likely, as we did for (6), this also means that our estimate is the most likely configuration.

**Theorem 1.** *We use the same abbreviations as in Section 3.1. Let  $p(X)$  be constant for all possible values of  $X$ . Then for any given  $|A|, |B|, a, b, x$ , choosing  $X$  to maximize*

$$p(a, b, x | X)$$

*also maximizes*

$$p(X | a, b, x)$$

*making  $X$  the most likely configuration.*

*Proof.* By standard probability theory we have

$$p(X | a, b, x) = \frac{p(X)}{p(a, b, x)} \cdot p(a, b, x | X)$$

Since we assumed  $p(X)$  to be constant, and  $p(a, b, x)$  is independent of  $X$ , the two expressions are related through a constant factor. Thus maximizing one maximizes the other as well.  $\square$

We can find the maximum for  $p(a, b, x|X)$  as follows. If for some value  $X$  we have

$$p(a, b, x|X) = p(a, b, x|X+1)$$

then the maximum lies between  $X$  and  $X+1$ , very close to  $X+0.5$ . Using equation (7), and the natural continuation of the discrete probability function  $p(a, b, x|X)$ , we can transform this into a quadratic equation and solve it accordingly (which is not hard but quite tedious, and thus omitted):

**Theorem 2.** *The equation*

$$p(a, b, x|X) = p(a, b, x|X+1)$$

*can be solved for  $X$  as follows:*

$$\begin{aligned} C_1 &:= x \cdot (|A| + |B| + 1 - (a + b + x)) \\ C_2 &:= |A| \cdot b \cdot (|A| - a) + |B| \cdot a \cdot (|B| - b) \\ C_3 &:= |A| \cdot b + |B| \cdot a - a \cdot b \\ a_0 &:= |A| \cdot |B| \cdot C_1 - C_2 \\ a_1 &:= C_3 - C_2 - (|A| + |B|) \cdot C_1 \\ a_2 &:= C_1 + C_3 \\ X &:= \frac{-a_1 \pm \sqrt{a_1^2 - 4 \cdot a_0 \cdot a_2}}{2 \cdot a_2} \end{aligned}$$

The correct solution is then the one with  $\pm$  being ‘-’ in the last formula, and adding 0.5 gives us an accurate value for the maximum. While this is somewhat more complex than the resemblance or subset containment based estimates that we will discuss in Section 3.3, it is more accurate and can be computed more efficiently (in constant time) than minimizing the standard error. Thus, as mentioned in Section 3.1, it can be used as a starting point to approximating  $\phi_{min}$  by integrating over a smaller interval of  $p(a, b, x|X)$  for uniform distributions. For non-uniform distributions, which we will discuss in Section 4, it is useful in estimating the distribution from an initial sample.

### 3.3 Related Work

As far as we know, there exists little prior work on estimating intersection size from samples. In Broder (1997) the author suggests to estimate the *resemblance*, that is

$$\rho = \frac{|A \cap B|}{|A \cup B|}$$

using the resemblance of their signatures  $S(A), S(B)$ :

$$\rho \approx \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|}$$

Knowing the set resemblance we could then compute the intersection size as

$$|A \cap B| = \frac{\rho}{\rho + 1}(|A| + |B|) \approx x \cdot \frac{|A| + |B|}{|S(A)| + |S(B)|}$$

which is indeed the approach taken in Dasu et al. (2002). However, this can be problematic when the ratio between set size and signature size varies, as the following example demonstrates.

**Example 1.** Let  $A, B$  be sets with  $|A| = 100, |B| = 200$  and a sampling result of  $a = 20, b = 10, x = 10$ . Its probability distribution is shown in Figure 1. Using resemblance we then estimate

$$|A \cap B| = x \cdot \frac{|A| + |B|}{|S(A)| + |S(B)|} = 10 \cdot \frac{300}{50} = 60$$

which is clearly not a good estimate.

A different estimation, also introduced by Broder (1997) to capture set containment is the following:

$$\frac{|A \cap B|}{|A|} \approx \frac{|S(A) \cap S(B)|}{|S(A)|}$$

This directly translates into an estimate for intersection size:

$$|A \cap B| \approx x \cdot \frac{|A|}{|S(A)|}$$

For Example 1 this formula provides an estimate of  $|A \cap B| \approx 33$ , which is better than the resemblance-based estimate but still far from perfect. Worse though, this changes to  $|A \cap B| \approx 100$  if we swap the roles of  $A$  and  $B$ . Even if we assume  $A$  to always be the smaller set we can get poor results though, especially if  $A$  and  $B$  are (roughly) of the same size.

**Example 2.** Let  $A, B$  be sets with  $|A| = 100, |B| = 110, a = 0, b = 20, x = 20$ . The corresponding probability distribution is shown in Figure 2. Then we would estimate

$$|A \cap B| \approx 20 \cdot \frac{100}{20} = 100$$

Clearly this figure is too large since  $B$  contains at least 20 elements not in  $A$ , so  $|A \cap B| \leq 90$ .

Note that such extremely poor estimates aren't very likely - in Example 1 the (effective!) sample set  $S(A)$  is bigger than  $S(B)$  even though  $A$  is only half as large as  $B$ , while in Example 2 the set  $S(B)$  is twice as large as  $S(A)$  even though  $B$  is only slightly larger than  $A$ . Still, they contribute to the expected (standard) error, making resemblance and subset containment based estimates much less accurate than necessary. We will compare the different estimation functions in Section 5.

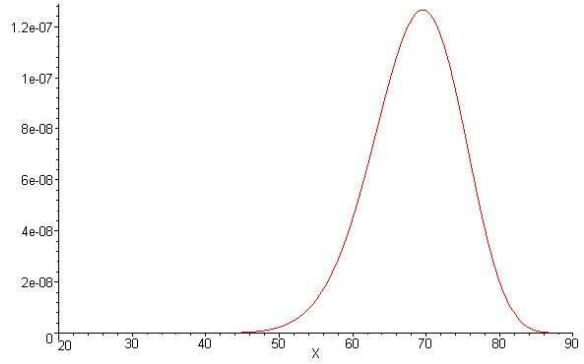


Figure 2: distribution of  $p(a, b, x|X)$  for  $|A| = 100, |B| = 110, a = 0, b = 20, x = 20$

One nice property which both the resemblance and subset containment based estimates display, is that they are unbiased. This result is shown by Broder (1997) for resemblance and subset containment, and extends to the derived estimation functions for intersection size as well. In contrast, neither our maximum likelihood estimate nor  $\phi_{min}$  minimizing the standard error are unbiased (although the bias for maximum likelihood is quite small). However, there appears to be no obvious need for estimates to be unbiased in our scenario. The main practical benefit of this property is that we can take multiple independent estimates and average then, and the result is then guaranteed to converge against the true value. In our case though, taking multiple independent samples increases total sample size, and is less accurate than simply taking a single larger sample (except for min hashing, where results are exactly the same).

### 4 Non-Uniform Distributions

While the assumption that all intersection sizes are equally likely is a “good guess” if we don't know anything about the real distribution and leads to reasonably simple formulas, it isn't optimal if the actual distribution is non-uniform. On the other hand, if we know the real distribution, we can incorporate it into our estimation formulas. Also, if we have such knowledge at sampling time, we can adjust the sample size to the minimum necessary to obtain the accuracy we require. It is worth noting that for non-uniform distributions, the necessary sample size can be much lower than for uniform ones. Consider e.g. the extreme case where sets are always either identical or disjoint - here a sample size of one is fully sufficient to obtain exact results. For distributions with only near-identical or near-disjoint sets, slightly bigger samples may be required, but still much less than for uniform distributions.

In some cases, probability distributions can be obtained by theoretical argument. More commonly though, it is initially unknown. In such cases, we can approximate it through a pre-sampling phase. Here we can employ the same sampling algorithms as for the main sampling phase, but with a larger sample size (sufficiently large to ensure that our estimates are accurate). However, to avoid high costs - after all, there is no point of having small samples if we first have to use large samples as well - we don't sample all sets but only some of them. This can provide us with a sufficiently accurate estimate for the probability distribution, which we then use in estimating intersection size for all (pairs of) sets.

While sampling only provides us with a discreet probability distribution, a continuous (or less ragged) distribution can easily



If such an approach is deemed too costly (e.g. due to technical/administrative overheads in sample generation), it is also possible to use two phases in estimating intersection sizes, using the same sample sets for both. In the first phase, we assume a uniform distribution, and use our results as estimate for the probability distribution of intersection sizes. In the second phase, we then use this distribution to make our estimates more accurate.

In principal, it is possible to repeat this process arbitrarily many times, but we found that results converge so quickly that one iteration is always sufficient. Also, in the first sampling phase, it is preferable to use an unbiased estimator even if it is less accurate, as we aggregate the results of multiple estimates. We found that the maximum likelihood estimate work very well here - while it is biased, the actual bias it displays is almost negligible, and it offers fast and accurate results.

For computing the optimal  $X$  which minimizes the standard error under a non-uniform distribution, we only need to use the actual (or approximated) value for  $p(X)$  in equation (5), which has almost no impact on computation time (provided that  $p(X)$  is pre-computed or that computation is fast). The maximum likelihood method is not generally applicable, as

$$\begin{aligned} p(X | a, b, x) &= \frac{p(X)}{p(a, b, x)} \cdot p(a, b, x | X) \\ &= \text{const} \cdot p(X) \cdot p(a, b, x | X) \end{aligned}$$

may have multiple peaks, and the highest peak isn't necessarily close to the optimal answer. To optimize our  $\phi_{min}$  approximation by integrating over a smaller interval, can first compute the intervals where  $p(a, b, x | X)$  and  $p(X)$ , respectively, exceed a threshold, and then integrate over the intersection of these intervals. We note that the maximum likelihood method may still be applicable to certain (e.g. gaussian) distributions if these are known in advance, but a careful study is necessary for each case.

## 5 Comparison of Estimation Functions

We will now evaluate and compare the estimation functions discussed in terms of accuracy and (briefly) efficiency, that is

$$\begin{aligned} \phi_{max} &:= X \text{ maximizing } p(a, b, x | X) \\ \phi_{res} &:= x \cdot \frac{|A| + |B|}{|S(A)| + |S(B)|} \\ \phi_{conA} &:= x \cdot \frac{|A|}{|S(A)|} \\ \phi_{conB} &:= x \cdot \frac{|B|}{|S(B)|} \\ \phi_{min} &:= (6) \end{aligned}$$

### 5.1 Accuracy

For comparison of accuracy, we will fix  $|A|, |B|$  and  $n := |S(A) \cup S(B)|$ , and assume a uniform distribution for  $X$ , i.e.,

$$p(X) = \frac{1}{\min(|A|, |B|) + 1}$$

for all  $X = 0 \dots \min(|A|, |B|)$ . Note that it is not necessary to evaluate this experimentally - we can easily be obtained using function smoothing techniques.

obtain exact results for the standard error by considering all possible cases for  $a, b, x, X$  and their probability to arise, as described in equation (3). A naive evaluation is expensive though. In particular, note that  $\phi(a, b, x)$  is evaluated multiple times for different  $X$ . While this could be avoided by storing results, we can actually do better:

**Theorem 3.** Consider equation (4):

$$\begin{aligned} Err(\phi, a, b, x) &:= (4) \\ &= \sum_{X=x}^{\min(|A|-a, |B|-b)} p(X) \cdot p(a, b, x | X) \cdot (\phi(a, b, x) - X)^2 \end{aligned}$$

Let  $\phi_{min}(a, b, x)$  be defined as in (5) and recall

$$p(a, b, x) = \sum_{X=x}^{\min(|A|-a, |B|-b)} p(X) \cdot p(a, b, x | X)$$

Then for any estimation function  $\phi$  we have

$$\begin{aligned} Err(\phi, a, b, x) &= Err(\phi_{min}, a, b, x) + \\ &= p(a, b, x) \cdot (\phi_{min}(a, b, x) - \phi(a, b, x))^2 \end{aligned}$$

Theorem 3 allows us to compute standard errors efficiently using  $\phi_{min}$  and the corresponding minimal standard error, making an exact evaluation feasible (for set sizes up to a few thousand - for larger sizes, approximations as discussed in section 3.1 become necessary, and approximated results for large sizes are simply scaled versions of exact results for smaller sizes).

Another interesting observation following directly from Theorem 3, is that any estimation function  $\phi$  can be regarded as trying to approximate  $\phi_{min}$ . The increase in quadratic standard error over the unavoidable error (the quadratic standard error for  $\phi_{min}$ ) is then simply the  $p(a, b, x)$ -weighted  $L^2$ -distance between  $\phi$  and  $\phi_{min}$ .

Graphs comparing the standard errors of the different estimation functions for different set and sample sizes are shown in Figures 3, 4 and 5.

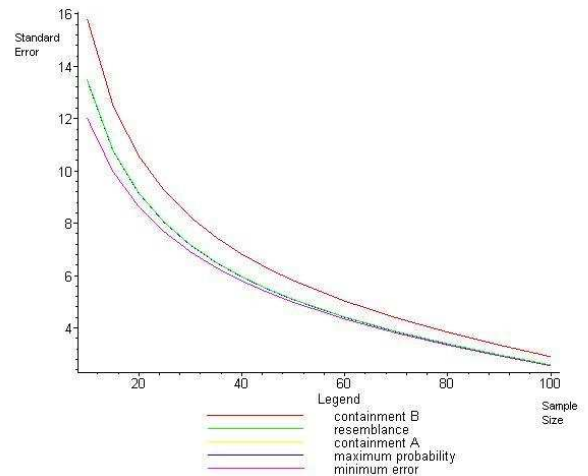
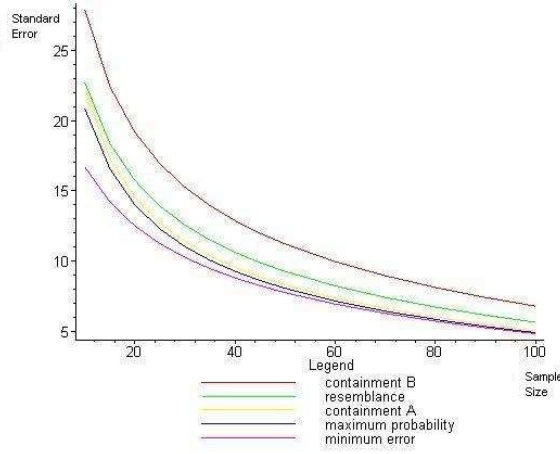


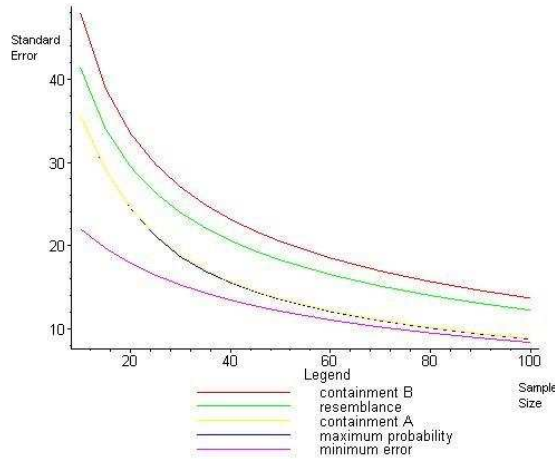
Figure 3: standard error for  $|A| = 100, |B| = 100$

For all figures, the horizontal axis denotes the total effective sample size  $|S(A) \cup S(B)|$ , while the vertical axis denotes the standard error. For equally sized sets  $A, B$  (Figure 3) we find that all approximations are reasonably accurate even for small sample sizes. Using  $\phi_{min}$  improves accuracy slightly compared  $\phi_{max}$  and  $\phi_{res}$  (which are equally accurate), although the

Figure 4: standard error for  $|A| = 100, |B| = 200$ 

difference quickly diminishes as sample size grows.  $\phi_{conA}$  and  $\phi_{conB}$  are identical in this case, but clearly less accurate than the other estimates.

When the size of  $A$  and  $B$  differs, as shown in Figures 4 and 5, we first note that  $\phi_{conA}$  becomes more accurate than  $\phi_{res}$  which in turn is more accurate than  $\phi_{conB}$ . This is rather intuitive, as using the smaller set  $A$  for the estimate prevents errors from growing too large. Also, the standard error as well as the difference in standard error between the estimation functions grows, thus increasing the benefit of using the more expensive estimate  $\phi_{min}$ .

Figure 5: standard error for  $|A| = 100, |B| = 500$ 

These trends continue as the size ratio between  $A$  and  $B$  increases. For  $|A| = 100, |B| = 500$  (and larger ratios) the estimates  $\phi_{conA}$  and  $\phi_{max}$  become equally accurate, while  $\phi_{res}, \phi_{conB}$  perform very poorly. By using  $\phi_{min}$  instead of the commonly employed  $\phi_{res}$ , we can reduce the sample size from 100 to about 25 while maintaining the same accuracy.

Note that this type of evaluation is possible for any estimation function  $\phi$ . While  $\phi_{min}$  cannot be improved in terms of accuracy, it may be possible to find close approximations which allow faster computation.

## 5.2 Computation

Computation of  $\phi_{min}$  has a time complexity of  $O(\min(|A|, |B|))$ , which is a direct consequence of (6) and (7). Using approximations as discussed in Section 3.1 can reduce this to an (average) complexity of  $O(S(A) \cup S(B))$  (for fixed  $k$ ). Thus computation is efficient for small samples, but not for very large

ones. Additionally, we have seen in the previous section that for large samples the difference between estimation functions shrinks. We therefore consider our approach most useful for small sample sizes (say, below 100), and less effective for larger ones. This means that it works best for applications with low accuracy demands (where, say, a 10% error may still be acceptable), as for high accuracy requirements larger sample sizes are unavoidable.

In our experiments, we found that computation of  $\phi_{min}$  took about  $10^2 - 10^3$  times longer than  $\phi_{res}, \phi_{conA}, \phi_{conB}$  and about  $10 - 10^2$  times longer than  $\phi_{max}$ , depending on sample size. Typically this is still much faster than computation of intersection sizes of samples though, and thus not a real concern in practice.

## 6 Conclusion

We have introduced a novel approach for estimating intersection size of sets from samples, based on a careful probability-theoretic examination. We demonstrated that there does indeed exist an optimal estimation function  $\phi_{min}$  which cannot be beaten by *any* other estimate, and that the accuracy of any function estimating intersection size can be judged by how well it approximates  $\phi_{min}$ . While exact computation of  $\phi_{min}$  can become too expensive for large sets, we provided effective approximations for it. Comparing our methods to existing approaches, we found that we can increase accuracy significantly (at an acceptable increase in computation time), or reversely achieve the same accuracy using smaller samples. As our analysis shows, this approach is most useful for small samples (up to a size of about 100), as for larger samples the gain in accuracy diminishes while computation time for our more accurate approximation increases.

For future work, it may be worth investigating how our approach can be applied to join size estimation. Also, the reverse problem of determining the minimal sample size needed to achieve a given accuracy deserves further study, and may become especially interesting for non-uniform distributions.

## References

- Bauckmann, J., Leser, U., Naumann, F. & Tietz, V. (2007), Efficiently detecting inclusion dependencies, in 'ICDE', pp. 1448–1450.
- Broder, A. (1997), On the resemblance and containment of documents, in 'SEQUENCES: Proceedings of the Compression and Complexity of Sequences', IEEE Computer Society, p. 21.
- Broder, A. Z. (2000), Identifying and filtering near-duplicate documents, in 'CPM', pp. 1–10.
- Broder, A. Z., Glassman, S. C., Manasse, M. S. & Zweig, G. (1997), 'Syntactic clustering of the web', *Computer Networks* **29**(8-13), 1157–1166.
- Chaudhuri, S., Motwani, R. & Narasayya, V. R. (1999), On random sampling over joins, in 'SIGMOD Conference', pp. 263–274.
- Dasu, T., Johnson, T., Muthukrishnan, S. & Shkapenyuk, V. (2002), Mining database structure; or, how to build a data quality browser, in 'SIGMOD Conference', pp. 240–251.
- Ganguly, S., Gibbons, P. B., Matias, Y. & Silberschatz, A. (1996), Bifocal sampling for skew-resistant join size estimation, in 'SIGMOD Conference', pp. 271–281.

- Haas, P. J. & Koenig, C. (2004), A bi-level bernoulli scheme for database sampling, *in* 'SIGMOD Conference', pp. 275–286.
- Manber, U. (1994), Finding similar files in a large file system, *in* 'USENIX Winter', pp. 1–10.
- Olken, F. & Rotem, D. (1986), Simple random sampling from relational databases, *in* 'VLDB', pp. 160–169.
- Rahm, E. & Bernstein, P. A. (2001), 'A survey of approaches to automatic schema matching', *VLDB J.* **10**(4), 334–350.



# A Batch Algorithm for Maintaining a Topological Order

David J. Pearce<sup>1</sup>Paul H.J. Kelly<sup>2</sup>

<sup>1</sup>School of Engineering and Computer Science, Victoria University of Wellington, NZ,  
Email: david.pearce@ecs.vuw.ac.nz

<sup>2</sup>Department of Computing, Imperial College London, UK, Email: p.kelly@imperial.ac.uk

## Abstract

The dynamic topological order problem is that of efficiently updating a topological order after some edge(s) are inserted into a graph. Much prior work exists on the unit-change version of this problem, where the order is updated after every single insertion. No previous (non-trivial) algorithms are known for the batch version of the problem, where the order is updated after every batch of insertions. We present the first such algorithm. This requires  $O(\min\{k \cdot (v+e), ve\})$  time to process any sequence of  $k$  insertion batches. This is achieved by only re-computing those region(s) of the order affected by the inserted edges. In many cases, our algorithm will only traverse small portions of the graph when processing a batch. We empirically evaluate our algorithm against previous algorithms for this problem, and find that it performs well when the batch size is sufficiently large.

## 1 Introduction

A topological order, *ord*, of a directed acyclic graph  $D = (V, E)$  maps each vertex to a priority value such that  $ord(x) < ord(y)$  holds for all edges  $(x, y) \in E$ . The *Dynamic Topological Order (DTO)* problem involves updating a topological order after an edge insertion. In the *unit-change* version of DTO, the topological order is updated after each edge insertion; in the *batch* version, the topological order is updated after each *batch* of insertions. The DTO problem has many applications and has been studied in the context of constraint-based pointer analysis [21, 9], compilation [14, 17], incremental evaluation of computational circuits [2], constraint-based local search algorithms [16], deadlock detection [4], machine-learning [26], and multiple sequence alignment [24, 25, 7]. For example, in constraint-based pointer analysis, topologically ordering vertices and detecting cycles in the (dynamically changing) constraint graph can dramatically reduce solving time [21, 18, 9]. Since many DTO algorithms (including that studied here) extend naturally to *dynamic cycle detection* [18, 11, 27], such algorithms are key to efficient pointer analysis. Furthermore, in this problem, edges are typically inserted in batches and, hence, there is much to gain from efficient solutions to the batch DTO problem.

Copyright ©2010, Australian Computer Society, Inc. This paper appeared at the Thirty-Third Australasian Computer Science Conference (ACSC2010), Brisbane, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 102, B. Mans and M. Reynolds, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

**Prior Work (Unit Change Problem).** In the unit-change problem, the topological order is updated after every edge insertion. A simple approach is to recompute the topological order from scratch after an edge insertion using a standard topological sort. Using a depth-first search, this yields an  $\Theta(v + e)$  time per insertion, where  $v = |V|$ ,  $e = |E|$ , since every vertex and edge is visited. We can easily improve upon this by first checking whether the inserted edge does, in fact, invalidate the current ordering. In some cases, it will not and, hence, we can avoid work. This gives an  $O(v + e)$  runtime, which reflects the fact that some insertions take  $\Theta(v + e)$  time, whilst others take  $\Theta(1)$  time.

Numerous works have built upon these basic principles to devise more efficient algorithms. In the majority of cases, efficiency is determined by considering an amortised bound on the time taken to process any sequence of  $k$  insertions. For example, the cost of inserting  $k$  edges using the simple approach outlined above is  $O(k \cdot (v + e))$  as, in the worst-case, the cost per insertion is  $O(v + e)$ .

One of the first works to improve upon this was the algorithm of Marchetti-Spaccamela *et al.* (henceforth, MNR) [15]. This processes a single edge insertion in  $O(v + e)$  time, and any sequence of  $k$  insertions in  $O(ve)$  time. Their algorithm is based upon the following observation:

**Definition 1.** Let  $D = (V, E)$  be a directed acyclic graph and *ord* a valid topological order. For an edge insertion  $(x, y)$ ,  $AR_{xy} = \{k \in V \mid ord(y) \leq ord(k) \leq ord(x)\}$ . We call  $AR_{xy}$  the Affected Region.

Marchetti-Spaccamela *et al.* showed that only vertices within the affected region need be repositioned to obtain a valid order. Furthermore, whilst the worst-case cost of processing a single insertion is the same as that of the simple approach outlined above, in practice it is very likely that MNR will do much less work as: firstly, both algorithms only do work when the inserted edge invalidates the ordering; secondly, MNR will only visit vertices and edges in the affected region, where as a standard topological sort will always visit *every* edge and vertex in the graph.

One of the most important works in this area is that of Alpern *et al.* [2]. They identified a lower bound,  $K_{min}$ , for the unit-change problem:

**Definition 2.** Let  $D = (V, E)$  be a directed acyclic graph and *ord* a valid topological order. For an edge insertion,  $x \rightarrow y$ , the set  $K$  of vertices is a cover if  $\forall a, b \in V. [a \rightsquigarrow b \wedge ord(b) < ord(a) \Rightarrow a \in K \vee b \in K]$ . A cover is minimal, written  $K_{min}$ , if it is not larger than any valid cover.

Alpern *et al.* provided an algorithm (henceforth, AHRSZ) whose runtime for a single edge insertion was bounded by the number of edges adjacent to members of  $K_{min}$ . However, they did not provide

an amortised bound on the time to process any sequence of  $k$  insertions. Zhou and Müller improved the space requirements of AHSZ [29]. Katriel and Bodlaender showed, for a slight variant of AHSZ, an  $O(\min\{k^{3/2} \log v, k^{3/2} + v^2 \log v\})$  bound on the time to insert  $k$  edges [12]. Liu and Chao obtained a tighter bound of  $O(k^{3/2} + kv^{1/2} \log v)$  for the Katriel-Bodlaender algorithm [13]. Kavitha and Mathew further improved this to  $O(k^{3/2} + k^{1/2}v \log v)$ . More recently, Haeupler *et al.* gave yet another variant on the AHSZ algorithm, and achieved an  $O(k^{3/2})$  bound on the time to process any sequence of  $k$  insertions [10].

Using a different approach, we developed a simpler algorithm (henceforth, PK) and experimentally showed it to be fastest on sparse random graphs [19, 20]. While this has inferior time complexity, compared with those based on the algorithm of Alpern *et al.*, it does have an important advantage: the AHSZ algorithm (and subsequent improvements) rely on an ordered-list data structure [8, 5] which suffers from high overheads in practice, and also from being rather difficult to implement. Ajwani *et al.* also took another approach and obtained an  $O(v^{2.75})$  bound with a different algorithm, thus improving upon the result of Katriel and Bodlaender for dense graphs [1]. Finally, Bender *et al.* very recently presented a new algorithm [6] which represents a radical departure from those before, in that it does not maintain an explicit ordering of vertices (which all previous algorithms do). For this, they obtained an  $O(v^2 \log v)$  time to process any sequence of  $k$  insertions, which improves upon all those before it, particularly for dense graphs.

**Prior Work (Batch Problem).** The batch version of the DTO problem is slightly more relaxed than the unit-change version. In this case, it is no longer required that the topological order be updated after *every* edge insertion; instead, edge insertions are packaged into batches, with each batch being processed in one go. Thus, the topological order must be updated after every batch of insertions. While this version of the problem arises in practical problems (see e.g. [18, 22, 23]), there have thus far been no specific solutions for it.

As before, a simple approach is to recompute the topological order from scratch after each batch  $B$  of insertions (see Algorithm 1). This yields an  $O(v + e)$  bound on the time to process a single batch  $B$  and, hence, takes  $O(k \cdot (v + e))$  time for a sequence of  $k$  insertion batches.

Another approach is to simply reuse one of the solutions to the unit-change problem. That is, to process a batch of  $b$  insertions as if it were a sequence of  $b$  individual insertions. Since, for each of the previous unit-change algorithms, the worst-case time for a single edge insertion is still  $O(v + e)$ , we arrive at a bound of  $O(\min\{kb \cdot (v + e), ?\})$  for a sequence of  $k$  insertion batches, each of which has at most  $b$  edges. The  $?$  in this bound is a cap on the total cost, as determined by the amortised bound obtained for the unit-change algorithm in question. For example, for MNR, the bound would be  $O(\min\{kb \cdot (v + e), ve\})$ , whilst for the algorithm of Bender *et al.* [6] it would be  $O(\min\{kb \cdot (v + e), v^2 \log v\})$ .

We can obtain an even better bound than this by combining both of these approaches together [11]. The idea is simply to run Algorithm 1 in parallel with one of the unit change algorithms when processing a sequence of insertion batches. Then, we simply see which one finishes first and use the topological order it produces (whilst stopping the other immediately). For example, let us consider using MNR here. Since Algorithm 1 takes at most  $O(k \cdot (v + e))$  for a sequence

---

**Algorithm 1** ADD\_EDGE( $B$ )

---

```

1: //  $B$  is a batch of updates
2: if  $\exists(x, y) \in B. [\text{ord}(y) < \text{ord}(x)]$  then
3:     perform standard topological sort

```

---

of  $k$  insertion batches, and MNR takes at most  $O(ve)$  for *any* insertion sequence, we arrive at a combined worst-case bound of  $O(\min\{k \cdot (v + e), ve\})$  for the parallel algorithm. This improves upon the worst-case bound of either algorithm in isolation. However, whilst this is certainly better in theory, it is also clear that it is not a particularly practical solution. In particular, there will be much redundant work performed by both algorithms as they are, in fact, operating in a very similar fashion. For example, both will perform depth-first traversals of the graph when, in fact, only one traversal is required. Thus, *one desires an algorithm which properly and efficiently combines Algorithm 1 and a unit-change algorithm whilst achieving the same bound without such redundancy.*

**Our Contribution.** At last, we can now discuss the contributions of this paper:

1. We present the first algorithm (henceforth, PK<sub>2</sub>) which genuinely integrates Algorithm 1 with a unit-change algorithm.
2. We provide a proof of correctness for algorithm PK<sub>2</sub>.
3. We present results from an empirical comparison of PK<sub>2</sub> against other unit-change algorithms. The results indicate the PK<sub>2</sub> outperforms the other algorithms when the batch size is sufficiently large.

Algorithm PK<sub>2</sub> does not suffer any of the redundancy inherent in the parallel algorithm discussed above. In particular, *it never traverses an edge or visits a vertex more than once when processing a batch of edge insertions.* It will also process a single edge in worst-case  $O(v + e)$  time, and any sequence of  $k$  batches in worst-case  $O(\min\{ve, k \cdot (v + e)\})$  time. To achieve this, we build upon algorithm MNR, primarily because it is the simplest of the unit-change algorithms. *Nevertheless, the bound we obtain on the time to process  $k$  batches is still better than all previous algorithms, except for the parallel algorithm discussed already.* For example, the best unit-change algorithm, due to Bender *et al.* [6], requires  $O(v^2 \log v)$  to process a single batch. This is a log factor worse than for our algorithm on dense graphs (i.e. when  $e = O(v^2)$ ), and will be more for sparse graphs (i.e. when  $e < O(v^2)$ ). Finally, we hope that this work will motivate future investigation into the batch DTO problem, which has so far been ignored by others.

## 2 Algorithm PK<sub>2</sub>

Before presenting algorithm PK<sub>2</sub>, we will first review the operation of algorithm MNR, upon which PK<sub>2</sub> is based.

### 2.1 Overview of MNR

The algorithm of Marchetti-Spaccamela *et al.* [15] employs an array of size  $|V|$  which maps each vertex to a unique integer from  $\{1 \dots |V|\}$ . In addition, a second array  $\text{ord}^{-1}$  of size  $|V|$  is used, which is the inverse of  $\text{ord}$  — it maps each index in the order to the corresponding vertex. For an invalidating edge  $(x, y)$ ,

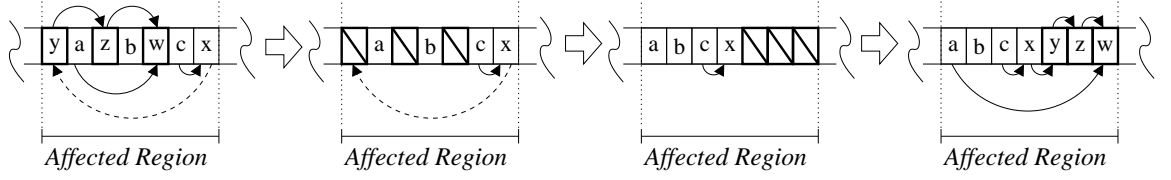


Figure 1: Illustrating the MNR algorithm updating a topological order. The original topological order is on the far left, while that obtained from running MNR is on the far right; in all cases, vertices are laid out in topological order (i.e. increasing in *ord* value) from left to right (hence, the invalidating edge is that drawn with a dashed line). In between, we see the two stages of the MNR algorithm: *discovery* and *shifting*. The former identifies vertices which are out-of-order after the edge insertion, and is implemented using a DFS from  $y$  (restricted to the affected region). The latter shifts those vertices found during discovery up the order, so that they now come after  $x$  whilst still retaining their original relative orders.

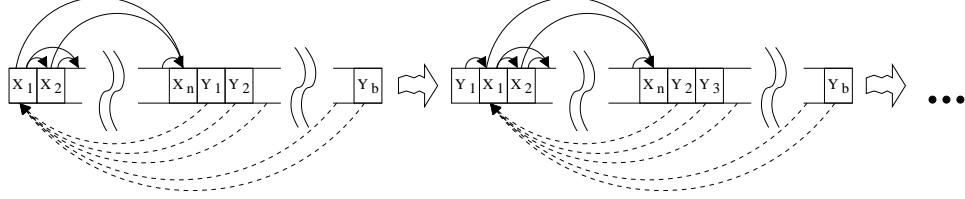


Figure 2: Illustrating a worst-case input for algorithm MNR (left), where the dashed arcs represent the edges to be inserted. The result of processing the first edge insertion  $(Y_1, X_1)$  is shown on the right. Here,  $n$  is  $\Theta(v)$  and there exist  $\Theta(e)$  edges of the form  $(X_i, X_j)$ , where  $i < j$  and  $X_1 \rightsquigarrow X_i$ . The sequence of new insertions contains  $b$  edges of the form  $(Y_i, X_1)$ . Assuming these are processed in order, starting from  $(Y_1, X_1)$ , then each requires  $\Theta(v + e)$  time since every edge reachable from  $X_1$  is traversed by the depth-first search. Thus, MNR needs  $\Theta(b(v + e))$  time to solve this graph.

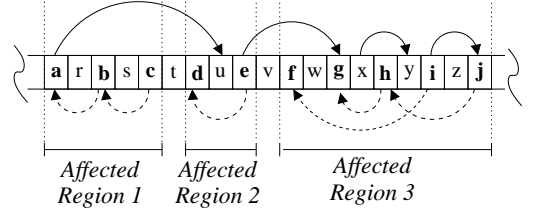
MNR identifies and removes nodes reachable from  $y$  in the affected region using a depth-first search (known as *discovery*). Then, it traverses the affected region from the bottom, shifting vacant spaces to the top. Nodes previously removed are now placed in their original order back into the vacant slots. Figure 1 illustrates this. MNR requires  $O(v + e)$  time to process an edge insertion. The worst-case occurs when the affected region includes  $\Theta(v)$  nodes and  $\Theta(e)$  edges reachable from  $y$ . Figure 2 illustrates the first edge,  $(Y_1, X_1)$ , being processed in a worst-case sequence for MNR. In fact, Marchetti-Spaccamela *et al.* obtained an  $O(v e)$  bound on the total time to process any sequence of insertions for MNR [15]. This caps the total cost of processing a batch; hence, even if  $b$  was  $\Theta(v^2)$  above (e.g. by allowing those of the form  $(Y_i, X_j)$ ), the runtime would not be  $\Theta(v^2 e)$  as might be expected. The proof of this relies on a simple observation that, if an edge  $v \rightarrow w$  is traversed as a result of an invalidating edge  $x \rightarrow y$ , then it won't be traversed again for any other invalidating edge whose tail is  $x$ . This is because, having processed  $x \rightarrow y$ , we have  $\text{ord}(x) \leq \text{ord}(v) < \text{ord}(w)$  and, for any subsequent invalidating edge  $x \rightarrow z$ , we have  $\text{ord}(z) \leq \text{ord}(x)$  (otherwise, it isn't invalidating).

## 2.2 Overview of PK<sub>2</sub>

We now present our new algorithm, referred to as PK<sub>2</sub> (since we refer to our earlier algorithm as PK [20]), for the batch DTO problem. The algorithm essentially extends MNR to the batch problem and, when the batch size is 1, they operate in an identical fashion. As with MNR, algorithm PK<sub>2</sub> employs two arrays, *ord* and *ord*<sup>-1</sup>, to map nodes to indices and vice-versa.

The key feature of algorithm PK<sub>2</sub> is that it *never visits or shifts a node more than once* when inserting a batch of edges (unlike MNR). To achieve this, we must alter our notion of the affected region so that overlapping regions are treated as one — so, although

a batch of insertions can still define several affected regions, they are all disjoint and can be processed independently. The following aims to clarify this:



Here, each affected region can be correctly ordered independently of the others, by rearranging its contents. Thus, we extend the definition of an affected region to a batch of overlapping edges by combining their affected regions as follows:

**Definition 3.** Let  $D = (V, E)$  be a DAG and *ord* a valid topological order. For a set  $B$  of overlapping, invalidating edge insertions, the *Affected Region* is denoted  $AR_B$  and defined as  $\{k \in V \mid b \leq \text{ord}(k) \leq t\}$ , where  $b$  (resp.  $t$ ) is the lowest (resp. highest) index of  $\{x \mid (x, y) \in B \vee (y, x) \in B\}$ .

## 2.3 Shift Procedure

The first difficulty lies in rearranging an individual affected region without visiting or shifting any node twice. To achieve this goal, we introduce the notion of a *shift set* as follows:

**Definition 4.** A *frontier pair*,  $(x, d)$ , is a pair of nodes in  $AR_B$  where  $d \rightsquigarrow x$ ,  $\text{ord}(x) < \text{ord}(d)$  and where  $\neg \exists z \in AR_B. [z \rightsquigarrow x \wedge \text{ord}(d) < \text{ord}(z)]$ . We refer to  $d$  as the destination of  $x$ .

Informally, a frontier pair  $(x, d)$  identifies a vertex  $x$  to be reordered and its destination  $d$ , which is the vertex furthest up the ordering where  $d \rightsquigarrow x$ . Our algorithm must ensure  $x$  is located above  $d$  in the final ordering.

---

**Algorithm 2** SHIFT( $i, Q$ )      //  $i$  is leftmost position in affected region,  $Q$  is a shift queue

---

```

1:  $n = 0$       // number of nodes temporarily removed from order so far
2: while  $Q \neq \emptyset$  do
3:    $w = \text{ord}^{-1}(i)$       //  $w$  is node at topological index  $i$ 
4:   if  $\text{vacant}(w)$  then
5:      $n = n + 1$  ;  $\text{vacant}(w) = \text{false}$ ;      // reset vacant flag as slot will be occupied by end
6:   else
7:      $\text{allocate}(w, i - n)$ 
8:     // now insert all nodes associated with index  $i$ 
9:      $(v, d) = \text{head}(Q)$ 
10:    while  $Q \neq \emptyset \wedge w = d$  do
11:       $n = n - 1$  ;  $\text{allocate}(v, i - n)$  ;  $\text{pop}(Q)$  ;  $(v, d) = \text{head}(Q)$ 
12:     $i = i + 1$ 

```

**procedure**  $\text{allocate}(v, i)$

```

12:  $\text{ord}(v) = i$  ;  $\text{ord}^{-1}(i) = v$       // place  $v$  at index  $i$ 

```

---

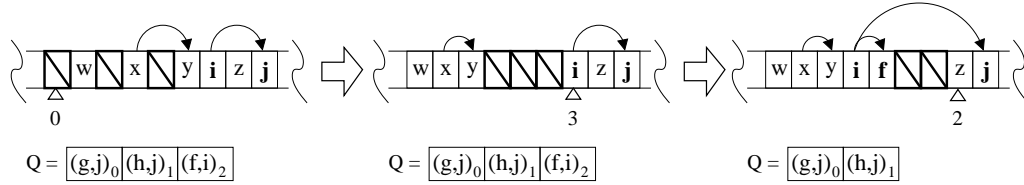


Figure 3: Illustrating the shift procedure of Algorithm 2 operating on affected region 3 from our running example. The topological order after discovery is shown on the left, where the spaces are slots vacated by nodes now on the shift queue  $Q$ . In the middle diagram, the algorithm has begun shifting vertices into the correct position; it has reached the destination of the head element on  $Q$ . In the final diagram, we see that  $f$  has been placed after its destination  $i$ , and the algorithm will proceed to push the vacant spaces up the order

**Definition 5.** For a set  $B$  of overlapping, invalidating edge insertions, the shift set is defined as the set of all frontier pairs  $(x, d)$  in  $AR_B$ .

For example, the shift set for Affected Region 3 in the example above (as  $j \rightsquigarrow i$ ) is:  $\{(f, i), (g, j), (h, j)\}$ . We can obtain a valid ordering from this by shifting each node immediately right of its destination, whilst topologically sorting those with the same destination (Lemma 1). To formalise this process, we refine our notion of a shift set into that of a *shift queue*:

**Definition 6.** A shift queue,  $Q$ , is a shift set whose members are also totally ordered. More specifically,  $\forall (x_1, d_1)_i, (x_2, d_2)_j \in Q. [(ord(d_1) > ord(d_2) \vee (d_1 = d_2 \wedge x_1 \rightsquigarrow x_2)) \Rightarrow i < j]$ . The head of the queue (i.e. the pair to be removed first) is that with the highest index.

The shift process operates by scanning the affected region from bottom-to-top whilst shifting vacant slots up the order. During this process, if the current vertex being examined is  $d$  and  $(v, d)$  is the head of the shift queue, then  $v$  is placed into the vacant slot immediately after  $d$ ; and, if the next element on the shift queue has the same destination, it is placed immediately after that and so on, until all those with the same destination are placed.

The shift procedure is detailed in Algorithm 2. Figure 3 illustrates the algorithm operating on Affected Region 3 from our example before, assuming the shift queue has been constructed already. Note, for clarity, the value of  $i$  on Line 2 is indicated by the triangular marker, whilst the value of  $n$  at that point is given below it. In the figure, the left diagram shows the spaces vacated by those now on the shift queue  $Q$ . In the middle diagram, the algorithm has begun shifting them up the order and has reached the destination of the head element on  $Q$ . In the final diagram, we see that  $f$  was placed after its destination  $i$ , and that the algorithm is proceeding to shift the remaining vacant spaces up the order.

**Lemma 1.** Let  $D = (V, E)$  be a DAG,  $\text{ord}$  a valid topological order (with  $\text{ord}^{-1}$  as its reverse map) and  $B$  a set of overlapping, invalidating edge insertions. Given  $Q$ , a shift queue for  $B$ , Algorithm 2 produces a valid topological order.

*Proof.* Let  $\text{ord}'$  be the updated topological order. Suppose  $\text{ord}'$  is invalid. Then, there is some  $x, y \in V$  where  $x \rightsquigarrow y$  and  $\text{ord}'(y) < \text{ord}'(x)$ . Since Algorithm 2 only repositions nodes within  $AR_B$ , it follows that  $x, y \in AR_B$  (otherwise,  $\text{ord}$  was invalid to begin with). There are five cases to consider:

- i)  $(y, d_y)_i \notin Q$  and  $(x, d_x)_j \notin Q$ . In this case, the relative positions of  $x$  and  $y$  are preserved by Algorithm 2 and, hence,  $\text{ord}'(y) < \text{ord}'(x) \Rightarrow \text{ord}(y) < \text{ord}(x)$  which is a contradiction.
- ii)  $(y, d_y)_i \in Q$  and  $(x, d_x)_j \notin Q$ . Here,  $\text{ord}(d_y) < \text{ord}(x)$  is needed to get  $\text{ord}'(y) < \text{ord}'(x)$  as  $y$  is placed immediately after  $d_y$  and the relative positions of  $d_y$  and  $x$  do not change (note,  $(d_y, dd_y) \notin Q$  follows from Definition 5). However, this implies  $Q$  is malformed under Definition 5 (as  $x \rightsquigarrow y \wedge \text{ord}(d_y) < \text{ord}(x)$ ).
- iii)  $(y, d_y)_i \notin Q$  and  $(x, d_x)_j \in Q$ . In this case,  $\text{ord}(d_x) < \text{ord}(y)$  follows from Definition 5 (otherwise,  $d_x$  would be  $y$ 's destination since  $x \rightsquigarrow y$ ). Again, since  $x$  is placed immediately after  $d_x$  and the relative positions of  $d_x$  and  $y$  do not change, we arrive at  $\text{ord}'(x) < \text{ord}'(y)$ .
- iv)  $(y, d_{xy})_i \in Q$  and  $(x, d_{xy})_j \in Q$ . Here,  $i < j$  follows from Definition 6 and, since vertices are placed in the order popped from  $Q$  and that with highest index is popped first, this gives  $\text{ord}'(x) < \text{ord}'(y)$ .

```

1:  $Q = \emptyset$  ;  $\text{sort}(B)$  // sort invalidating edges into descending order by index of tail
2: for all  $i = 0 \dots |B|$  do
3:    $(x, y) = B[i]$ 
4:   if  $\neg \text{vacant}(y)$  then  $\text{dfs}(y, \text{ord}(x))$ 
5: return  $Q$ 

procedure  $\text{dfs}(v, ub)$ 
6:  $\text{vacant}(v) = \text{true}$  ;  $\text{onStack}(v) = \text{true}$ 
7: for all  $(v, s) \in E$  do
8:   if  $\text{onStack}(s)$  then abort // cycle detected
9:   if  $\neg \text{vacant}(s) \wedge \text{ord}(s) < ub$  then  $\text{dfs}(s, ub)$  // visit if not already and in  $AR_B$ 
10:  $\text{onStack}(v) = \text{false}$  ;  $\text{push}((v, \text{ord}^{-1}(ub)), Q)$ 

```

**Algorithm 4** ADD\_EDGE( $B$ )      //  $B$  is a set of edge insertions

---

```

1:  $E = E \cup B$ ;
2: for all  $(x, y) \in B$  do
3:   if  $\text{ord}(x) < \text{ord}(y)$  then  $B = B - \{(x, y)\}$  // remove forward edges from  $B$ 
4: if  $|B| > 0$  then
5:    $\text{sort}(B)$  // sort invalidating edges into descending order by index of tail
6:    $lb = |V|$  // lowerbound of current region
7:    $s = 0$  // start of current region
8:   for all  $i = 0 \dots |B| - 1$  do
9:      $(x, y) = B[i]$ 
10:    if  $\text{ord}(x) < lb \wedge i \neq 0$  then
11:       $Q = \text{DISCOVER}(\{B[s], \dots, B[i-1]\})$ ;  $\text{SHIFT}(lb, Q)$ 
12:       $s = i$  // start of new region
13:       $lb = \min(\text{ord}(y), lb)$ ;
14:    // Process final region
     $Q = \text{DISCOVER}(\{B[s], \dots, B[|B| - 1]\})$ ;  $\text{SHIFT}(lb, Q)$ 

```

---

**Lemma 3.** Assume  $D = (V, E)$  is a DAG and  $\text{ord}$  an array mapping each vertex to a unique index from  $\{1 \dots |V|\}$ , with  $\text{ord}^{-1}$  as its reverse map. If a batch  $B$  of overlapping, invalidating edge insertions introduces a cycle, Algorithm 3 will detect this and abort.

*Proof.* Suppose  $B$  introduces a cycle  $C$ , and  $\text{dfs}(x, i)$  is first call involving a vertex  $x \in C$ . Algorithm 3 processes invalidating edges with highest tail first, hence  $\forall w \in C. [\text{ord}(w) \leq i]$ . Thus,  $\text{dfs}(x, i)$  will continue visiting vertices of  $C$  until it finds one where  $\text{onStack} = \text{true}$  (which must exist as  $C$  is a cycle) — at which point it aborts.  $\square$

### 2.5 Putting it All Together

Pseudo-code for the complete algorithm  $\text{PK}_2$  is given in Algorithm 4. This identifies distinct affected regions and processes them using Algorithms 2 and 3. As such, the overall runtime for a single edge insertion is  $O(v + e)$  which follows from the individual runtimes of Algorithms 2 and 3. For any sequence of  $k$  insertions, we obtain the following amortised bound:

**Theorem 1.** Let  $D = (V, E)$  be a DAG and  $\text{ord}$  a valid topological order (with  $\text{ord}^{-1}$  as its reverse map). Then,  $\text{PK}_2$  requires  $O(\min\{k \cdot (v + e), ve\})$  time to process any sequence of edge insertions split into  $k$  batches, where  $e$  is the number of edges in the final graph.

*Proof.* Let the insertion sequence be split into batches  $B_1, \dots, B_k$ . Suppose an edge  $(v, w)$  is traversed whilst processing some batch  $B_i$ . This can only occur if there is an  $(x, y) \in B_i$ , where  $y \rightsquigarrow v$ . Furthermore,  $(v, w)$  is traversed exactly once whilst processing  $B_i$  (this follows because nodes are visited according to a depth-first search). Now, after processing  $B_i$  is completed,  $\text{ord}(y) < \text{ord}(v)$  will hold. It follows that  $(v, w)$  will not be traversed again as a result of any insertion  $(y, z)$  (for any  $z$ ) occurring after this point. This is because the affected region for such an edge must extend to the left of  $y$ , but  $v$  will always be right of  $y$  (from now on). Thus, an edge can be traversed at most  $v$  times during any sequence of insertions. Furthermore, since no edge can be traversed more than once whilst processing a batch, an edge can be traversed at most  $k$  times when processing  $k$  batches.  $\square$

Theorem 1 is a straightforward extension to the proof originally given by Marchetti-Spaccamela *et al.* to show algorithm MNR runs in  $O(ve)$  time for any sequence of edge insertions [15]. This bound improves upon the  $O(\min\{kb \cdot (v + e), ve\})$  bound obtained by MNR for a sequence of  $k$  batches containing at most  $b$  edges and, in fact, over all other previously known

algorithms for this problem (except for the parallel algorithm discussed in §1). Furthermore, while  $\text{PK}_2$  does the same amount of work as Algorithm 1 in the worst case, there are many situations where  $\text{PK}_2$  does much less. This is because for an invalidating insertion, Algorithm 1 always visits every node and every edge, whereas  $\text{PK}_2$  visits only those within an affected region. This difference is highlighted by the following:

### 3 Experimental Study

In this section, we experimentally compare the performance of algorithm  $\text{PK}_2$  against various algorithms for this problem: STS (recall Algorithm 1), MNR [15], PK [19] and AHRSZ [2]. As with MNR, neither PK nor AHRSZ offers any benefit to processing edges in batches, rather than one at a time. The experiments measure how the *Average Cost Per Insertion (ACPI)* varies with batch size at different graph densities, over a large number of randomly generated DAGs.

**Definition 7.** For a DAG with  $v$  nodes and  $e$  edges, define its density to be  $\frac{e}{\frac{1}{2}v(v-1)}$ . Thus, it is the ratio of the number of actual edges to the maximum possible.

**Definition 8.** The model  $G_{\text{dag}}(v, p)$  is a probability space containing all graphs having a vertex set  $V = \{1, 2, \dots, v\}$  and an edge set  $E \subseteq \{(i, j) \mid i < j\}$ . Each edge of such a graph exists with a probability  $p$  independently of the others.

The model  $G_{\text{dag}}(v, p)$  was first defined by Barak and Erdős [3]. Using this, a DAG with  $v$  nodes and expected density  $x$  can be generated by setting  $p = x$ . Our experiments determined the Average Cost Per Insertion (ACPI) for each algorithm by measuring the time taken to insert a sample of edges into a DAG whilst maintaining a topological order. To do this, we generated 100 random DAGs with 2500 vertices at density 0.001, and 100 random DAGs with 2500 vertices at density 0.01. The edge set for each graph was divided into those making up the graph itself and those making up the insertion sample. The size of the insertion sample was fixed at 360 edges to ensure the total amount of work done remained constant across all experiments. For a given algorithm and batch size  $b$ , the average time taken to process the insertion sample in batches of  $b$  edges was recorded for each graph. An important point is that the insertion sample may include non-invalidating edges and these dilute our measurements, since the algorithms do no work for these cases. Our purpose, however, was to determine what performance can be expected in practice, where it is unlikely all edge insertions will be invalidating.

All experiments were performed on a 900Mhz Athlon based machine with 1GB of main memory, running Mandrake Linux 10.2. The executables were compiled using gcc 3.4.3, with optimisation level “-O3” and timing was performed using the `gettimeofday` function, which gives microsecond resolution. To reduce interference, experiments were performed with all non-essential system daemons/services (e.g. X windows, `crond`) disabled and no other user-level programs running. The implementation itself was in C++ and took the form of an extension to the *Boost Graph Library* v1.33.0, utilising the `adjacency_list` class to represent a DAG [28]. The complete implementation, including C++ code for all three algorithms and the random graph generator, is available online at <http://www.ecs.vuw.ac.nz/~djp>.

**Figure 4** shows the results of our experiments comparing ACPI for PK<sub>2</sub>, MNR, STS, PK and AHRSZ across varying batch sizes at densities 0.001 and 0.01. The plots for MNR, PK and AHRSZ are flat since they obtain no advantage from processing edges in batches. We see that PK<sub>2</sub> is always a better choice than either MNR or STS and, in many cases, offers significant gains over them. When the batch size is one, little difference is observed between MNR and PK<sub>2</sub> which reflects their close relationship. At density 0.01, the gap between these two algorithms has reduced. This is because, on dense graphs, there are few invalidating edges in the insertion batch as the graph is already highly ordered (hence, most insertions are not invalidating — see [20] for more on this). Thus, there is less chance the affected regions for two invalidating edges will overlap (as there are simply fewer affected regions) which is needed for PK<sub>2</sub> to obtain an advantage over MNR. While PK<sub>2</sub> is the clear winner at density 0.01, compared with PK and AHRSZ, it is less conclusive on the sparser graphs. Certainly, on small batches, it does not perform well by comparison. This is not surprising, since MNR performs poorly on sparse graphs as well and, on small batches, PK<sub>2</sub> and MNR will have similar behaviour. Indeed, given the gains obtained by PK<sub>2</sub> over MNR (on which it is based), it seems quite clear that extending either PK or AHRSZ to deal with batch insertions more efficiently would be valuable. The data also indicates that, for large batches, the performance of STS approaches that of PK<sub>2</sub>. This is expected as it becomes highly likely here that most nodes will be a member of some affected region and, hence, will be reordered by PK<sub>2</sub> anyway. Of course, PK<sub>2</sub> can still obtain an advantage because it does not always need to traverse every edge (as STS does).

**Figure 5** shows the results of a second experiment which measured the number of vertices and edges visited or shifted by each algorithm, rather than ACPI (note, all other experimental parameters remain the same). This is useful as it gives us a clear picture regarding the amount of work being performed by each algorithm, which is not muddled by the performance characteristics of the experimental machine. The charts show a striking resemblance to those of Figure 4 and give a strong indication that the results of Figure 4 are not dependent on the experimental machine, rather it is a direct function of the underlying algorithm.

**Limitations.** The results indicate that PK<sub>2</sub> is always an improvement upon MNR, and that it provides a useful improvement over the other algorithms in certain situations. However, we have not been

able to compare PK<sub>2</sub> against all prior unit-change algorithms (primarily because of the difficulty in implementing these algorithms). However, most of the other unit-change algorithms (i.e. [29, 12, 13, 10]) are minor variants on algorithm AHRSZ, and we would expect them to have very similar performance in practice. The remaining algorithms are that of Ajwani *et al.* [1] and Bender *et al.* [6]. In their paper, Ajwani *et al.* experimentally compare their algorithm against PK, MNR and AHRSZ and find that it only offers an improvement on a particular (artificially constructed) hard class of input graphs. Thus, we would not expect to see their algorithm performing better than PK in any of our experiments. Finally, then, remains the algorithm of Bender *et al.* which is very recent, and does yield the best amortised bound for any sequence of  $k$  edge insertions. Bender *et al.* do not report any experimental results for their algorithm, and it remains unclear how efficient it will be in practice. Nevertheless, it would be nice if this algorithm could be empirically evaluated in the future.

## 4 Conclusion

We have presented the first DTO algorithm which requires  $O(\min\{k \cdot (v+e), ve\})$  time to process any sequence of  $k$  edge insertion batches. We have experimentally evaluated it against various previous algorithms for this problem, demonstrating that: first, it always outperforms a standard topological sort and the related MNR algorithm; secondly, that it is generally better than the others when the batch size is large enough.

**Acknowledgements.** Thanks go to Irit Katriel, Gary Haggard and some anonymous referees for helpful comments on earlier drafts of this paper.

## References

- [1] D. Ajwani, T. Friedrich, and U. Meyer. An  $\tilde{O}(n^{2.75})$  algorithm for online topological ordering. In *Proc. Scandinavian Workshop on Algorithm Theory*, volume 4059 of *LNCS*, pages 53–63. Springer-Verlag, 2006.
- [2] B. Alpern, R. Hoover, B. K. Rosen, P. F. Sweeney, and F. K. Zadeck. Incremental evaluation of computational circuits. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 32–42. ACM Press, 1990.
- [3] A. Barak and P. Erdős. On the maximal number of strongly independent vertices in a random acyclic directed graph. 5(4):508–514, 1984.
- [4] F. Belik. An efficient deadlock avoidance technique. *IEEE Transactions on Computers*, 39, 1990.
- [5] M. A. Bender, R. Cole, E. D. Demaine, M. Farach-Colton, and J. Zito. Two simplified algorithms for maintaining order in a list. In *Proceedings of the European Symposium on Algorithms (ESA)*, volume 2461 of *Lecture Notes in Computer Science*, pages 152–164. Springer-Verlag, Sept. 2002.
- [6] M. A. Bender, J. T. Fineman, and S. Gilbert. A new approach to incremental topological ordering. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1108–1115. SIAM, 2009.

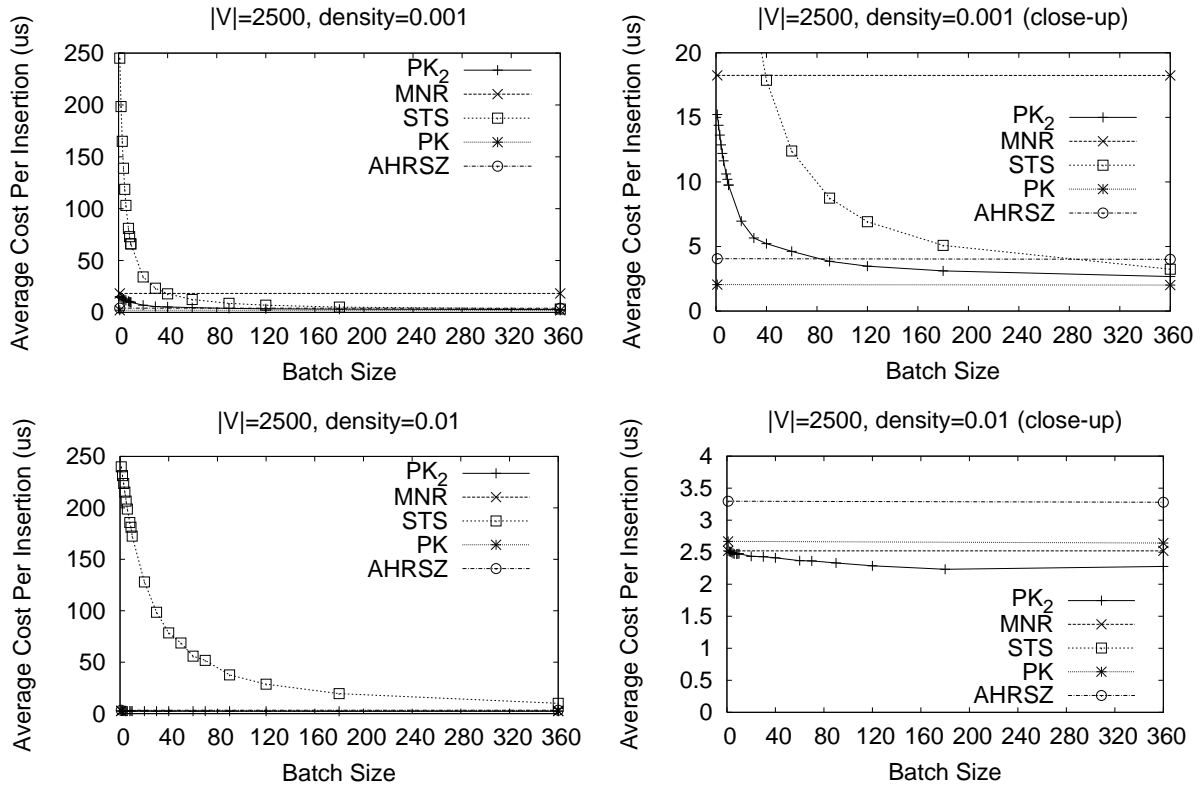


Figure 4: Experimental results looking at the effect of increasing batch size for all three algorithms on random DAGs with 2500 nodes at densities 0.001 and 0.01. For each, batch size is plotted against ACPI and we provide close ups at each density to capture interesting features.

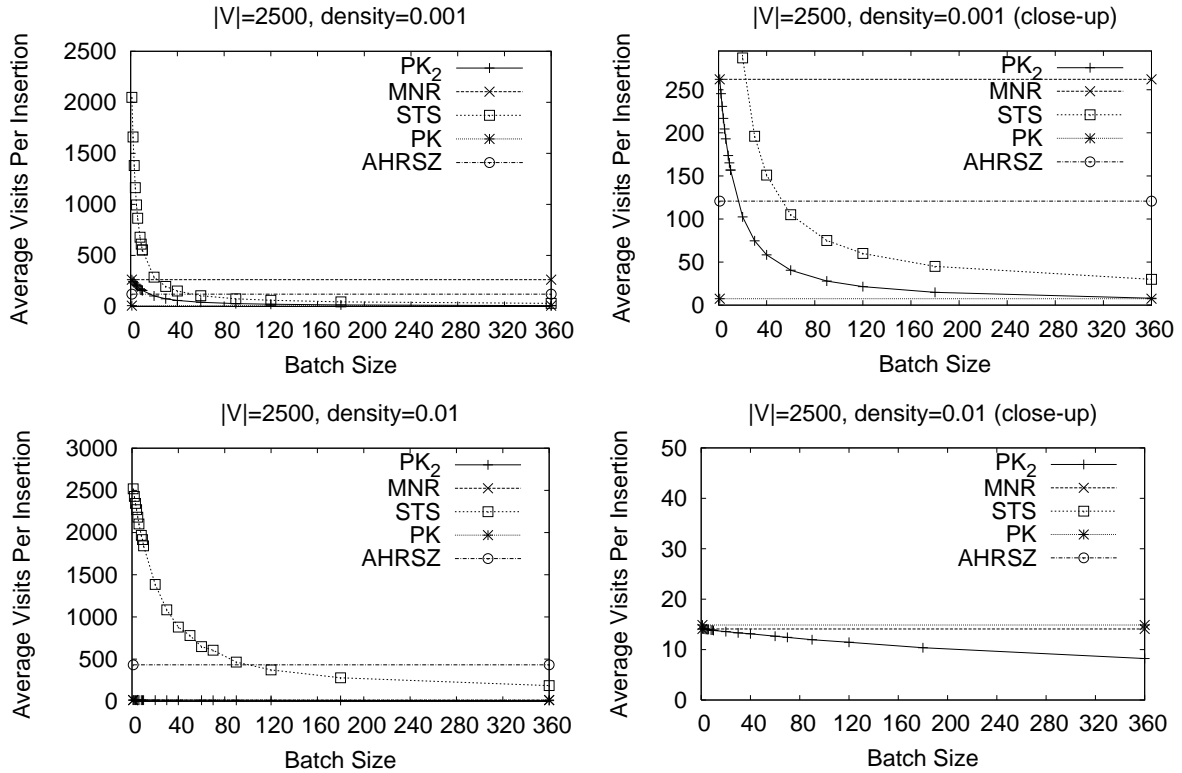


Figure 5: Experimental results looking at the effect of increasing batch size for all three algorithms on random DAGs with 2500 nodes at densities 0.001 and 0.01. For each, batch size is plotted against the average number of nodes and edges visited or shifted when processing an edge insertion. We provide close ups at each density to capture interesting features.



- [7] R. K. Bradley, L. Pachter, and I. Holmes. Specific alignment of structured RNA: stochastic grammars and sequence annealing. *Bioinformatics*, 24(23):2677–2683, 2008.
- [8] P. F. Dietz and D. D. Sleator. Two algorithms for maintaining order in a list. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 365–372. ACM Press, May 1987.
- [9] M. Fähdndrich, J. S. Foster, Z. Su, and A. Aiken. Partial online cycle elimination in inclusion constraint graphs. In *Proc. conference on Programming Language Design and Implementation*, pages 85–96. ACM Press, 1998.
- [10] B. Haeupler, T. Kavitha, R. Mathew, S. Sen, and R. E. Tarjan. Faster algorithms for incremental topological ordering. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 5125 of *Lecture Notes in Computer Science*, pages 421–433. Springer, 2008.
- [11] B. Haeupler, T. Kavitha, R. Mathew, S. Sen, and R. E. Tarjan. Incremental cycle detection, topological ordering, and strong component maintenance. Technical report, Department of Computer Science, Princeton University, 2008.
- [12] I. Katriel and H. L. Bodlaender. Online topological ordering. In *Proc. ACM Symposium on Discrete Algorithms*, pages 443–450. ACM Press, 2005.
- [13] H.-F. Liu and K.-M. Chao. An  $\tilde{O}(n^{2.5})$ -time algorithm for online topological ordering. *Theoretical Computer Science*, 389(1-2):182–189, 2007.
- [14] A. Marchetti-Spaccamela, U. Nanni, and H. Rohnert. On-line graph algorithms for incremental compilation. In *Workshop on Graph-Theoretic Concepts in Computer Science*, pages 70–86, 1993.
- [15] A. Marchetti-Spaccamela, U. Nanni, and H. Rohnert. Maintaining a topological order under edge insertions. *Information Processing Letters*, 59(1):53–58, 1996.
- [16] L. Michel and P. V. Hentenryck. A constraint-based architecture for local search. In *Proc. Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 83–100. ACM Press, 2002.
- [17] S. Omohundro, C.-C. Lim, and J. Bilmes. The sather language compiler/debugger implementation. Technical report, International Computer Science Institute, Berkely, 1992.
- [18] D. J. Pearce. *Some directed graph algorithms and their application to pointer analysis*. PhD thesis, Imperial College, London, UK, 2005.
- [19] D. J. Pearce and P. H. J. Kelly. A dynamic algorithm for topologically sorting directed acyclic graphs. In *Proc. Workshop on Efficient and Experimental Algorithms*, volume 3059 of *LNCS*, pages 383–398. Springer-Verlag, 2004.
- [20] D. J. Pearce and P. H. J. Kelly. A dynamic topological sort algorithm for directed acyclic graphs. *ACM Journal of Experimental Algorithmics*, 11:1.7, 2007.
- [21] D. J. Pearce, P. H. J. Kelly, and C. Hankin. On-line cycle detection and difference propagation: Applications to pointer analysis. *Software Quality Journal*, 12(4):309–335, 2004.
- [22] D. J. Pearce, P. H. J. Kelly, and C. Hankin. Efficient field-sensitive pointer analysis for C. *ACM Trans. Prog. Lang. Syst.*, 30(1), 2008.
- [23] F. M. Q. Pereira and D. Berlin. Wave propagation and deep propagation for pointer analysis. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO)*, pages 126–135. IEEE Computer Society, 2009.
- [24] A. S. Schwartz. *Posterior Decoding Methods for Optimization and Accuracy Control of Multiple Alignments*. PhD thesis, University of California, Berkeley, 2007.
- [25] A. S. Schwartz and L. Pachter. Multiple alignment by sequence annealing. *Bioinformatics*, 23(2):24–29, 2007.
- [26] J. Shen, L. Li, and W.-K. Wong. Markov blanket feature selection for support vector machines. In D. Fox and C. P. Gomes, editors, *AAAI*, pages 696–701. AAAI Press, 2008.
- [27] O. Shmueli. Dynamic cycle detection. *Information Processing Letters*, 17(4):185–188, Nov. 1983.
- [28] J. Siek, L.-Q. Lee, and A. Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley, 2002.
- [29] J. Zhou and M. Müller. Depth-first discovery algorithm for incremental topological sorting of directed acyclic graphs. *Information Processing Letters*, 88(4):195–200, 2003.



# GeoWeight: Internet Host Geolocation Based on a Probability Model for Latency Measurements

M. J. Arif, S. Karunasekera

Department of Computer Science and Software Engineering  
University of Melbourne, Australia  
marif, shanika@csse.unimelb.edu.au

S. Kulkarni

NICTA Victoria Labs, Australia  
santosh.kulkarni@nicta.com.au

## Abstract

Knowing the geographical location of an Internet host is of importance to many of today's Internet services. In this paper we focus on geolocating Internet hosts based purely on latency measurements. Existing latency measurement-based geolocation techniques use the observed latencies from multiple landmarks to the target host to determine maximum bound or both the maximum and minimum bounds of the geographical region where the target host is located. Due to the large variance of Internet latency measurements, the region constrained based on such maximum-minimum bounds tends to be relatively large resulting in large estimation errors. We propose a geolocation algorithm, GeoWeight, which improves the geolocation accuracy by further limiting the possible target region by dividing the constrained region to sub-regions of different weights. The weight assigned to a sub-region indicates the probability of the target being in that sub-region; a higher weight indicating a more probable region. By considering latency measurements from multiple landmarks and computing the resultant weights of overlapping regions a better constrained target region can be obtained. This paper presents the GeoWeight algorithm and evaluates its performance using both synthetic and real data by geolocating target hosts in North America. We compare GeoWeight with two popular geolocation techniques, Octant and CBG, by geolocating the same set of targets. The results show that the GeoWeight algorithm outperforms existing techniques.

## 1 Introduction

Internet host geolocation is an important research problem that is currently addressed by many research groups. Internet location information can be leveraged to improve the user experience and determine business strategy. Some uses of such location-aware systems include geographically targeted advertising on web sites, automatic selection of language to display web site content, web content delivery based on region, credit card fraud detection, and load balancing and resource allocation between Internet hosts.

Our goal is to develop a scalable and reliable Geolocation technique to locate hosts on the Internet.

However, what makes this task challenging is there is no one-to-one mapping between IP addresses and geographic locations. The dynamic nature of IP address assignment makes the host geolocation in an IP environment even harder. On the other hand, wireless domain localization is well addressed [20], as the transmission characteristics in the air are relatively regular. Internet host localization is more difficult because the transmission characteristics on the Internet are abruptly influenced by factors such as circuitous route and queueing delay on the routers.

Current measurement-based approaches for geolocation mainly use end-to-end latency measurements from a set of nodes with known location to the node to be geolocated. Nodes with known locations are referred as *landmarks* and the nodes to be geolocated are referred to as *targets*. Based on the observed positive correlation between latency and distance travelled by data packets, these latency-based geolocation techniques constrain the estimated location of the target [18, 13, 23]. These approaches confine the region where the target is estimated to reside to within a maximum distance around each landmark. The use of a constraint for the minimum distance from the landmark, in addition to the maximum possible distance, is shown to improve the geolocation accuracy. Such positive and negative distance factors are developed based on the maximum and minimum bounds of the distance to latency relationship. However, the variability of latency measurements between Internet hosts yields a significant disparity between the positive and negative distance bounds. As a result, area to which the target is constrained using these methods is relatively large. Refining location information using additional geographical hints [18, 23] has shown to improve the geolocation accuracy. The integration of the underlying network topology information has been another method considered for improving Internet host geolocation [4].

In this paper, we present a novel geolocation algorithm, GeoWeight, which is based purely on Internet latency measurements. The GeoWeight algorithm accounts for the possible variability of distance (between the minimum and maximum possible distances) for a given latency by assigning weights to sub-regions within the region constrained by minimum and maximum bounds. The weights are assigned to sub-regions to reflect the probability that the target could be located in the respective subregion; a higher weight indicating more probable regions. The weights for the sub-regions are computed based on the Internet latency measurements for different distances as described in section 4. Latency measurements from multiple landmarks to the target result in intersect-

Copyright ©2010, Australian Computer Society, Inc. This paper appeared at the Thirty-Third Australasian Computer Science Conference (ACSC2010), Brisbane, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 102, B. Mans and M. Reynolds, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

ing regions. GeoWiegth algorithm computes a weight for an intersecting region as the sum of weights of overlapping regions enclosed in the intersection. The location of the target is chosen as the centroid of the intersection region having the highest computed weight. By assigning weights to sub-regions within the larger region, the GeoWeight algorithm is able to constrain the target location to a smaller region than that was possible from previous approaches, hence resulting in better estimation accuracies for geographical location.

In this paper we present the GeoWeight algorithm and also the technique for computing weights for different regions. We evaluate the performance of the proposed algorithm using simulated and real distance vs latency data. Through simulation, we specifically investigate two noise models for latency data, a Gamma distribution and a Lognormal distribution, to understand the impact of the noise model on the accuracy of the algorithm. We evaluate the performance of the algorithm by geolocating 60 hosts in North America. The weights for different regions in this case are computed based on large set of latency vs distance data we gathered over a month using 50 landmarks in North America using the PlatnetLab test bed. We compare our results by geolocating the same target hosts using two primarily-latency-based geolocation techniques [23, 13] and the results show that our technique outperforms both these techniques.

This paper discloses three main contributions. First, the paper develops a probability model for Internet latency. Second, it presents a novel measurement based geolocation approach, GeoWeight, for Internet host geolocation. Third, it evaluates the performance of GeoWeight and compares it with two existing measurement based approaches. The results show that GeoWeight outperforms the existing approaches in terms of *geolocation error*, distance between actual and estimated target locations.

The rest of the paper is organized as follows. Section 2 describes related work. The geolocation problem is formulated in Section 3. Internet latency data modeling and the GeoWeight algorithm are presented in Section 4. Section 5 evaluates the geolocation techniques using synthetic and measured data. Finally, conclusions are drawn and future directions are outlined in Section 6.

## 2 Related Work

Host Geolocation on the Internet is an important research problem that has been addressed by a number of research groups in the past. One of the intuitive approaches to host geolocation is a comprehensive IP tabulation against physical locations which can be used as a lookup table [16, 2]. However, because of the large number of available Internet hosts, such an approach will not scale. Also, a lookup table is difficult to maintain and keep up-to-date, especially, as it cannot take into consideration dynamic IP assignment.

Three techniques for geolocation were proposed in IP2Geo [18]: GeoTrack, GeoPing and GeoCluster. GeoTrack uses traceroute information from a host to the target, which contains the list of routers encountered along the path. Using location hints from the DNS names of the routers along the path, the locations of the routers are determined. Of the routers whose locations are known, the closest one to the target is selected, and its location is chosen as the target location. The accuracy of the technique depends on the distance from the target to the nearest router of known location. Next, GeoPing works on the assumption that hosts that are geographically close have sim-

ilar network delays with respect to other fixed hosts. By comparing the ping times to the target from a set of landmarks or probe machines with the ping times to a set of nodes at known locations, GeoPing estimates the target location to be the same as that of the node with known location having the most similar ping values. Thus, the accuracy of GeoPing is limited by the distance to the nearest probe. The third approach, GeoCluster, is a database lookup technique which groups IP addresses to clusters based on geographical proximity. This information is combined with the user registration database from web based services such as e-mail services. This technique suffers from the general problems related to database lookup-based approaches, such as reliability, scalability and maintainability issues and also unavailability of the user registration database for public access.

Recent data-mining based approach Structon [5] is similar to GeoCluster except that it uses publicly available web pages instead of proprietary data sources in order to extract geolocation information. Structon uses a three step approach. First, extracted geolocation information from web pages are associated with their IP addresses. Then, these mapping information goes through multi-stage inference processes in order to improve the accuracy and coverage of its IP geolocation repository of different IP segments. Finally, those IP segments that are not covered in the first two steps, are mapped with the location of the access router with the help of traceroute tool. The accuracy of Structon implementation on the Internet depends heavily on the accuracy of extracted geographical mapping information. Moreover, with Structon [5], it is harder to get accuracy more than in the granularity of city level.

Constraint-Based Geolocation (CBG) [13] uses ping times from landmarks as a measure of latency. For each landmark a maximum distance bound for a given latency is derived using distance-to-ping relationships observed between landmarks. During geolocation the observed latencies from landmarks to the target are used to draw circles centered at each landmark based on the maximum distance bounds derived earlier. The target is assumed to reside in the convex region resulting from the intersection of circles, and the target location is estimated as the centroid of this convex region. This technique requires the target to be geographically well surrounded by landmarks.

Similar to CBG, Topology-based Geolocation (TBG) [4] computes the possible location of the target as a convex region. In TBG, the maximum distance bound is obtained based on the maximum transmission speed of packets in fibre which gives a conservative estimate of the possible region. This region is further refined using inter-router latencies along the path from the target to the landmark, obtained from the traceroute command. The final target location is obtained through a global optimization that minimizes average position error for the target and the routers.

A more recently proposed measurement-based technique for geolocation is Octant [23]. In contrast to other constraint based approaches that only limit the area where the target may be located, Octant also identifies areas where the target may not be located based on observed latencies (referred to as negative constraint). Octant expresses such information by considering two circles corresponding to the maximum and minimum distances from each landmark to the target which constrains the possible geographical area where the target may be located. Each landmark fits a convex hull to all of its delay-to-distance data points with other landmarks. Upper and lower facets of the convex hull correspond to the maximum and minimum distance bounds. Different weights are

assigned to different geographical areas based on the number of intersections (higher weights assigned to larger numbers of intersections). The final estimated region is the union of all regions, where the weight exceeds a desired weight or the region size exceeds a selected threshold. A Monte-Carlo algorithm is applied to pick the best single point location from the final estimated regions. These estimated regions in Octant often end up being disconnected parts. In contrast, it is highly unlikely with GeoWeight. As in GeoWeight the maximum (positive) and minimum (negative) distance bounds are divided into different weighted regions. Octant uses geographical and demographical constraints to improve the localization accuracy beyond its measurement-only solution.

In addition to the above cited references [10, 17, 24, 12] also discuss Internet host geolocation. Other relevant research which includes geographic properties of routing [21], delay prediction or distance estimation between Internet nodes [6, 22, 15, 11, 19] and exploring nearby servers [14] have also contributed to the area of host geolocation.

GeoWeight differs from other measurement based approaches because it uses a weighted model for latency-to-distance measurements to estimate the target location.

### 3 Problem Formulation

This section presents the problem statement for GeoWeight.

#### 3.1 Problem Statement

The problem considered here is the geolocation of a target  $H$ . Let us denote the unknown position  $P_0$  of the target in terms of its latitude and longitude ( $lat_0, lon_0$ ).

Suppose that  $\{L_1, L_2, \dots, L_N\}$  be a set of  $N$  landmarks. Let  $(lat_i, lon_i)$  be the latitude and longitude of the  $i$ th landmark  $L_i$ . We carry out geolocation using latency measurements from the  $N$  landmarks to the target. Let  $\mathbf{t}_i = \{t_{i,j}\}_{j=1}^{n_i}$  be the set of  $n_i$  latency measurements from landmark  $i$  to the target. We denote the cumulative set of all measurements from landmarks to the host by:  $\mathbf{t}_{1:N} = \{\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_N\}$ .

Our goal is to estimate the location  $P_0$  of target  $H$  using measurements  $\mathbf{t}_{1:N}$ . We denote the estimated location of the target by  $\hat{P}_0$ .

Then the *geolocation error*  $\epsilon$  is defined as,

$$\epsilon = dist(P_0, \hat{P}_0) \quad (1)$$

where  $dist(P_x, P_y)$  represents the geographical distance between position  $P_x$  and  $P_y$ .

#### 3.2 The Latency Model

Considering that Internet data packets in the majority of the cases travel through optical fibres, the minimum latency between two nodes that are a distance  $d$  apart can be given by,

$$t_{min}(d) = d/c_{fibre} \quad (2)$$

. Here  $c_{fibre}$  is the maximum transmission speed of data through the fibre, which is approximately  $2/3$  the speed of light [13]. As we show later, the latencies observed in real world Internet traffic are significantly higher than this lower bound due to factors such as router congestion. Thus, the observed latencies are modeled as,

$$t(d) = t_{min}(d) + E(d) \quad (3)$$

where  $t(d)$  is the observed latency for distance  $d$  and  $E(d)$  is a noise term that accounts for network delays in the real measurement.

## 4 The GeoWeight Algorithm

This section presents the GeoWeight algorithm.

### 4.1 Initial Observations

Figure 1(a) shows a plot of the distance vs latency (approximately 150,000 data points) gathered on the PlanetLab test bed using 50 PlanetLab nodes in North America as landmarks. The Internet Control Message Protocol (ICMP) [3] ping delay between landmarks was used as the measure of latency. More details of the experimental setup is described in section 5.2.1. The solid line below the data points in figure 1(a) shows  $t_{min}(d)$  given by equation 2. Figure 2 shows the histogram of observed distances for a given latency range (90.05 ms-100.00 ms). In order to ascertain the observation of our PlanetLab dataset we also analyzed the dataset collected by iPlane [1] during same period. This dataset is based on latency measurements, shown in figure 1(b), between their 68 landmarks which similar to our landmarks are spread around North America.

Following are five characteristics observed from this data (figure 1(a) and 1(b)):

- The minimum latency observed is higher than the theoretical minimum given by the equation 2.
- There is a positive correlation between latency and distance.
- A simple linear or non-linear relationship is not apparent in the data set - the data is noisy as described by equation 3.
- Although an upper bound on distance for a given latency is apparent, a lower bound is not apparent. This is the case even for the data analyzed on a per landmark basis.
- For a given latency (or a latency range), some distances are more probable than other distances (Figure 2).

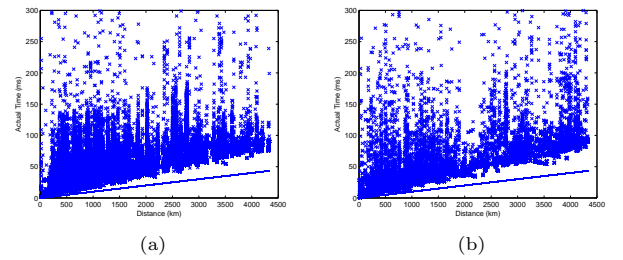


Figure 1: Distance-latency relationship (a)between 50 PlanetLab Landmarks (b)between iPlane dataset landmarks. The straight line below the data points shows delay to distance relationship according to equation 2

### 4.2 Existing Techniques

The current latency based approaches for target geolocation attempt to take into consideration the first four of the above observed characteristics of the distance to latency relationship.

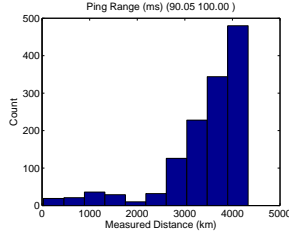


Figure 2: Histogram of distances for latency range 90.05-100.00 ms in the PlanetLab dataset

The CBG [13] technique uses the maximum distance bound in order to constrain the target location. Based on the observed latency from a landmark, the target location is constrained to the circular region around the landmark based on the upper bound of distance. Considering latency measures from multiple landmarks, the region of the target is considered as the convex region with maximum number of intersecting regions. The centroid of this convex region is estimated as target location. Figure 3 shows an example of the CBG approach for the case of a target being geolocated using three landmarks. As the figure shows, the CBG approach rightfully constrained the target inside the convex region based on the maximum distance bound. However, the drawback of this approach is the relatively large area the target is constrained to, around the landmark, due to considering only the maximum distance bound. This results in a relatively large final target region, hence potentially large geolocation errors.

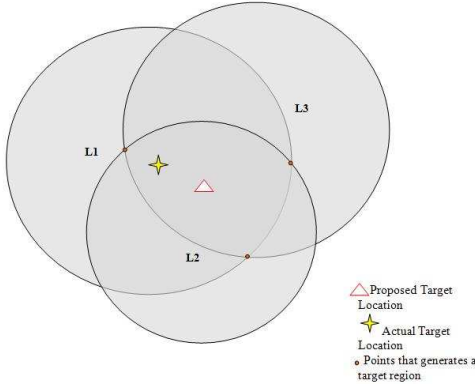


Figure 3: CBG Example

The Octant [23] approach improves the CBG approach by considering negative constraints. This technique defines the maximum-bound of distance as a positive constraint, within which the node must lie and a negative constraint, which indicates a minimum distance from the landmark where the target is constrained not to be present. Figure 4 shows an example of Octant approach for the case where a target is geolocated using three landmarks. Compared to the CBG approach, Octant reduces the possible target region size with the help of negative constraints. However, the latency vs distance measures we gathered show that the lower bound of distance for a given ping time is not apparent. As a result, constrained region produced by this technique could still be large resulting in high geolocation errors.

### 4.3 GeoWeight Approach

In addition to the first four characteristics of the distance to latency relationship identified in section 4.1,

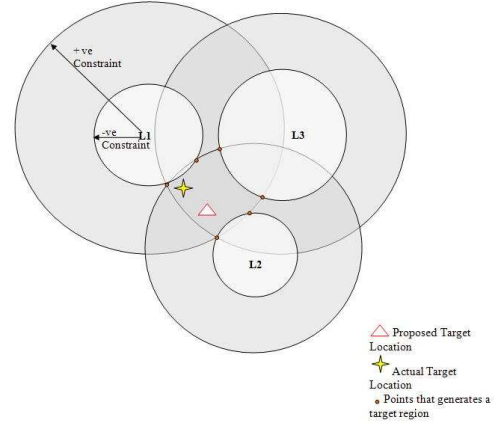


Figure 4: Octant Example

the GeoWeight algorithm takes into consideration the fifth characteristic of the distance-latency relationship; for a given latency, some distances are more probable than other distances. The GeoWeight algorithm uses this characteristic to constrain the possible region of the target to a smaller region than that was possible in the previous approaches as we describe below.

Let  $t_x$  be an observed latency from an arbitrary landmark. Based on the distance-latency relationship let  $d_x^{min}$  and  $d_x^{max}$  be the minimum and maximum possible distances for  $t_x$ . Consider the distance range from  $d_x^{min}$  to  $d_x^{max}$  is divided to  $N_{x,d}$  number of equal sized distance bins. The  $j$ -th bin, ( $j = 1, 2, \dots, N_{x,d}$ ), covers the distance range from  $d_{x,j}^{min}$  to  $d_{x,j}^{max}$  given by,

$$d_{x,j}^{min} = d_x^{min} + (j-1)(d_x^{max} - d_x^{min})/N_{x,d}$$

$$d_{x,j}^{max} = d_x^{min} + j(d_x^{max} - d_x^{min})/N_{x,d}$$

Let  $w_{x,j}$  be the weight for the  $j$ -th distance bin corresponding to  $t_x$ . The weight  $w_{x,j}$  represents the probability of the distance being in the range  $d_{x,j}^{min}$  and  $d_{x,j}^{max}$  for the observed latency  $t_x$ . The details of weight computation will be described in section 4.4.

For a given latency  $t_x$ , the GeoWeight algorithm considers  $N_{x,d}$  number of regions around the landmark, with the  $j$ -th region having distance bounds  $[d_{x,j}^{min}, d_{x,j}^{max}]$  and a weight of  $w_{x,j}$ . The latency measurements from multiple landmarks will result in intersecting regions, with different numbers of overlapping regions in each intersection region. The final weight for each intersection region is computed as the sum of weights of intersecting regions. The region of highest weight is considered as the constrained region of the target and the centroid of the region is estimated as the target location.

We simplify the generic algorithm presented above by considering the minimum distance, maximum distance and the number of distance bins to be the same for any latency, i.e.,

$$d_x^{min} = D_{min}$$

$$d_x^{max} = D_{max}$$

$$N_{x,d} = N_d$$

where  $N_d$  is the number of distance bins for any latency under consideration and  $D_{min}$  and  $D_{max}$  are the minimum and maximum possible distances for the geolocation scenario. For example, in section 5, where we evaluate our algorithm, the considered



Table 1: An example weight table for GeoWeight

Ping Time	0-250 (km)	250-500 (km)	500-750 (km)	750-1000 (km)
100	0	0	0.2	0.8
35	0.2	0.6	0.2	0
15	0.7	0.2	0.1	0

$D_{min}$  and  $D_{max}$  for the geolocation scenario are set such that it covers the whole of the North-American region. The only implication of the above simplification is some distance bins having a weight of 0 due to these distance ranges not being probable for the particular latency.

Figure 5 illustrates an example of the GeoWeight algorithm for a geolocation scenario with latency measurements from three landmarks. In this example the observed latency measures from the three landmarks L1, L2 and L3 are 100, 35 and 15 ms respectively. Table 1 shows the computed weights for the three latencies for different distance regions considering four ( $N_d = 4$ ) equidistant bins in the distance range 0-1000 km (in this example,  $D_{min} = 0$ ,  $D_{max} = 1000$ ).

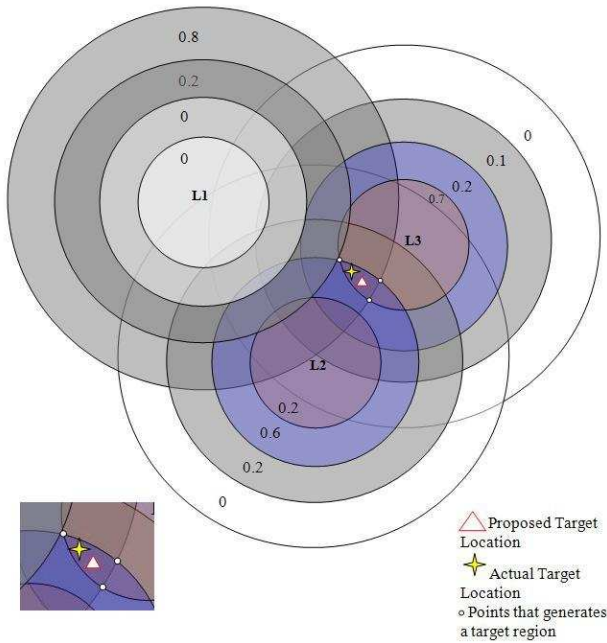


Figure 5: GeoWeight Example

Figure 5 shows the four regions ( $N_d = 4$ ) with different weights around each landmark in form of circles. These circles overlap with each other and the weight of each intersection region is computed as the sum of weights of the overlapping circles in the intersection region. For clarity, figure 5 does not show weights of all regions.

In this example, the region of maximum weight has a weight of 2.1 ( $0.8 + 0.6 + 0.7$ ) and the final target location is selected at the centroid of this region as shown in figure 5.

#### 4.4 Weight Computation

This section describes how the weights for the different distance regions are computed. As mentioned before, the weight for a distance range for a given latency is the probability of the distance being in the range for the given latency.

Consider a latency vs distance data set gathered from Internet measurements covering a distance range  $D_{min}$  to  $D_{max}$ . Let  $T_{min}$  and  $T_{max}$  be the minimum and maximum observed latencies. Consider this distance range and the time range divided to  $N_d$  and  $N_t$

Table 2: Weight computation example: the number of observed measurements for different time and distance ranges are shown in the table

	(0-500)km	(500-1000)km	(1000-1500)km	(1500-2000)km
(0-10)ms	100	25	0	0
(10-20)ms	15	120	35	0
(20-30)ms	12	52	95	12
(30-40)ms	5	23	126	32
(40-50)ms	2	10	24	68
(50-60)ms	0	5	12	128
(60-70)ms	0	12	21	45
<b>Total</b>	<b>134</b>	<b>247</b>	<b>313</b>	<b>285</b>

equidistant bins respectively. Let  $c_{ij}$  be the number of data point corresponding to the  $i$ -th time and  $j$ -th distance bin.

Table 2 shows an example of delay and distance bins. In this example,  $T_{min} = 0$ ,  $T_{max} = 70$ ,  $D_{min} = 0$ ,  $D_{max} = 2000$ ,  $N_d = 4$  and  $N_t = 7$ .

Each row represents the distance bins corresponding to a single time bin. Each column represents the time bins corresponding to a single distance bin. Each cell of the table shows the number of data points within a given time-distance bin.

Since the latency measures are collected for specific distances (i.e. inter landmark distances), even if the same number of latency measurements is gathered for each distance, the total number of distances represented in each distance bin will vary between distance bins as shown in table 2. Therefore, the first step in computing the weights is the normalization across distance bins by dividing the number in each distance-latency cell by the total number of measurements for the particular distance bin. The normalized distance-latency  $NR_{i,j}$  are given by,

$$NR_{i,j} = c_{i,j} / \sum_{i=1}^{N_t} c_{i,j} \quad (4)$$

The final weight for each cells is computed by normalizing across the latency bin, given by,

$$w_{i,j} = NR_{i,j} / \sum_{j=1}^{N_d} NR_{i,j} \quad (5)$$

where  $w_{i,j}$  is the weight of  $i$ th latency bin of  $j$ th distance bin which is the probability of the distance region  $[d_j^{min}, d_j^{max}]$  given the observed latency is in the range  $[t_i^{min}, t_i^{max}]$ .

In the example shown in table 2, considering the bin  $i = 1$  and  $j = 1$ ,  $NR_{1,1} = 100/134$  (0.74),  $NR_{1,2} = 25/247$  (0.10),  $NR_{1,3} = 0/313$  (0) and  $NR_{1,4} = 0/285$  (0). The sum of normalized weights  $S_1$ ,  $(0.74 + 0.10 + 0 + 0) = 0.84$ . Therefore  $w_{1,1} = NR_{1,1}/S_1 = 0.88$ ,  $w_{1,2} = NR_{1,2}/S_1 = 0.11$ ,  $w_{1,3} = NR_{1,3}/S_1 = 0$ ,  $w_{1,4} = NR_{1,4}/S_1 = 0$ .

## 5 Evaluation

This section presents the results of the experiments we conducted to evaluate our algorithm. We evaluated GeoWeight using simulated data as well as real Internet data. We also compared GeoWeight with two existing techniques; Octant and CBG. The results are presented in sections 5.1 and 5.2 respectively.

### 5.1 Evaluation based on Simulated Data

We conducted experiments using latency measurements simulated based on the latency model presented in section 3 using two different probability



models for noise. The aim of these experiments was to:

- Determine the optimum values of  $N_d$  and  $N_t$  for the algorithm.
- Evaluate the algorithm for known noise models.

The results of these experiments are presented in the following sections.

### 5.1.1 Experiment 1: Determining $N_d$ and $N_t$

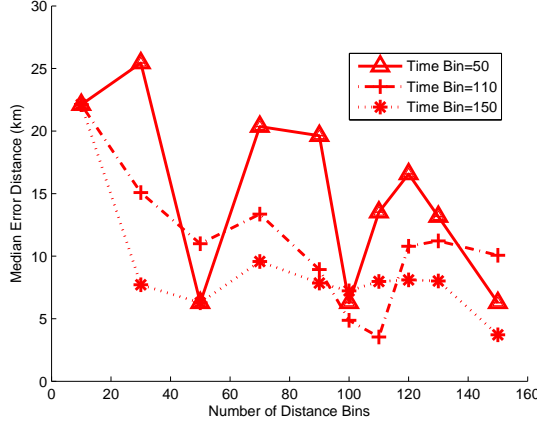


Figure 6: Performance of GeoWeight with different time and distance bins combination in noise free case

The GeoWeight algorithm considers finite size bins for distance and time for weight computation. As a result, when latency measures from multiple landmarks is considered, even for the noise free case, GeoWeight produces region of maximum weight as opposed to a single point. Therefore, the GeoWeight algorithm will not result in a zero geolocation error even for the noise free case, unlike distance based triangulation that would give a zero error. The accuracy of the GeoWeight algorithm will depend on bin size. Smaller bin sizes will result in lower errors provided the weight for the bin can be computed accurately. However, the accuracy of weight computation will be limited by the number of available measurements for a distance-time bin. Therefore the resolution of the distance bins will be determined by the number of distances in the data set. Similarly the number of data points for a distance bin will determine the resolution of the time bins. Thus, the goal of this experiment was to investigate the best possible bin sizes (determined by the number of time bins ( $N_t$ ) and number of distance bins ( $N_d$ )) to be used for computing the weights.

In this experiment, we use simulated latency data generated for the 1200 inter landmark distances in our data set with 1000 data points for each distance. We consider the noise free case (equation 2) where the distance to latency relationship is linear. We estimate the geolocation error ( $\epsilon$ ) by varying  $N_t$  and  $N_d$ . We geolocate 60 targets using 50 landmarks. The location of the landmarks and the targets are the same as our real landmarks and targets locations as described in section 5.2.1.

Figure 6 shows the median geolocation error results of three distinct values of  $N_t$  for  $N_d$  in the range of 10-150. The figure shows results of only three  $N_t$  for simplicity. As the figure shows  $[(N_t, N_d)=(110, 110)]$  combination resulted in the lowest median geolocation error distance. Therefore,  $[(N_t, N_d)=(110,$

110)] is used for weight computation in the subsequent experiments.

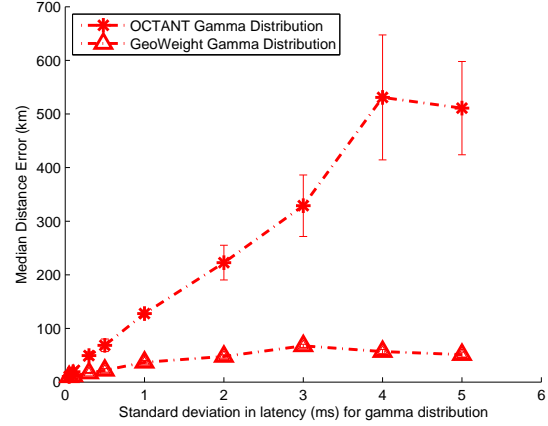


Figure 7: Median geolocation error as a function of variance for GeoWeight and Octant with Gamma noise distribution with mean = 5

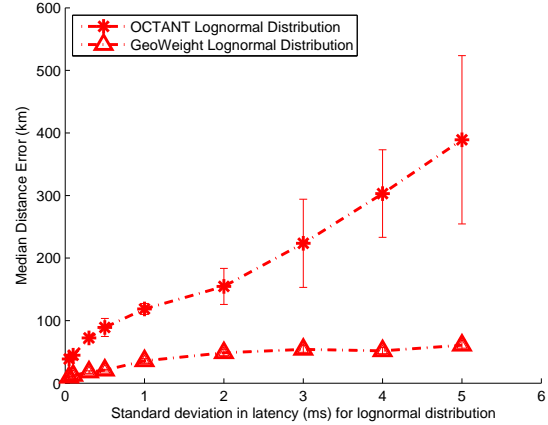


Figure 8: Median geolocation error as a function of variance for GeoWeight and Octant with Lognormal noise distribution with mean = 5

### 5.1.2 Experiment 2: Evaluation of GeoWeight

The goal of this experiment was to investigate the impact of different noise models on the geolocation error ( $\epsilon$ ). We also compare our algorithm with Octant's measurement-only solution for the different noise models.

In this experiment we investigated two different noise models: lognormal and gamma distribution. These two distributions were chosen because of their skewed and non-negative properties [8], similar to our real data set. Moreover, previous study [9] found lognormal characteristics in the Internet latency distribution. However, we acknowledge that these noise models do not accurately model the observed latencies. The experiments are to identify the behavior of the algorithm and understand the theoretical limits of accuracy of our algorithm.

The probability distribution function of the Gamma distribution is given by:

$$p(t|a, b) = \frac{1}{b^a \Gamma(a)} t^{a-1} \exp \frac{-t}{b}$$

where  $a, b$  are the parameters of the distribution. The mean  $m$  and the variance  $v$  are given by,

$$m = ab \quad (6)$$

$$v = ab^2 \quad (7)$$

The probability distribution function of the Lognormal distribution is given by:

$$p(t|\mu, \sigma) = \frac{1}{\sigma t \sqrt{2\pi}} \exp \frac{-(\ln(t) - \mu)^2}{2\sigma^2}$$

The mean  $m$  and the variance  $v$  are given by,

$$m = e^{\mu + \sigma^2/2} \quad (8)$$

$$v = (e^{\sigma^2} - 1)e^{2\mu + \sigma^2} \quad (9)$$

where  $\mu, \sigma$  are the mean and the standard deviation of the corresponding normal distribution.

We generated the weight matrix using simulated distance-latency data for the 1200 inter landmark distances using the noise model. Similar to experiment 1, we geolocated 60 targets using 50 landmarks to evaluate the performance of GeoWeight. The latency measures from the landmark to the target were also generated using the same noise model. We computed the median geolocation error by varying the mean and the variance of noise. We also computed the geolocation error of the Octant algorithm for these noise models. The measurement-only step of Octant algorithm was implemented using the algorithm details provided in [23].

Figure 7 and 8 show the median geolocation error for GeoWeight and Octant for different noise variances for Gamma and Lognormal noise distributions respectively. The mean value of noise is chosen to be 5 in each case. We have run the experiment for different values of mean but did not observe significant differences in the results for GeoWeight and Octant. Thus, for clarity figure 7 and 8 only show the case of  $m = 5$ .

For both noise models, it is evident that the median geolocation error increases with the increase in noise variance as expected. The rate of increase is higher for Octant than GeoWeight which shows that the geolocation error of Octant is more sensitive to noise than GeoWeight. This is due to the increase in the difference between the maximum and minimum distance bounds for higher noise variances. This results in a large bounding region for Octant whereas GeoWeight accommodates this difference by dividing the region to small regions of different weights.

### 5.1.3 Experiment 3: The impact of the number of landmarks

In this experiment we investigate the impact of the number of landmarks on the geolocation error distance  $\epsilon$ . In each experiment we select a random subset of the 50 landmarks, and we use this subset to geolocate the 60 targets.

Figure 9 shows median geolocation error as a function of the number of landmarks for Gamma and Lognormal noise distributions. The noise mean and variance were chosen to be 4 and 16 respectively. It can be seen that the geolocation error decrease with the increase in the number of landmarks. This is because higher number of landmarks allow to better constrain the target region since the intersection regions end up smaller.

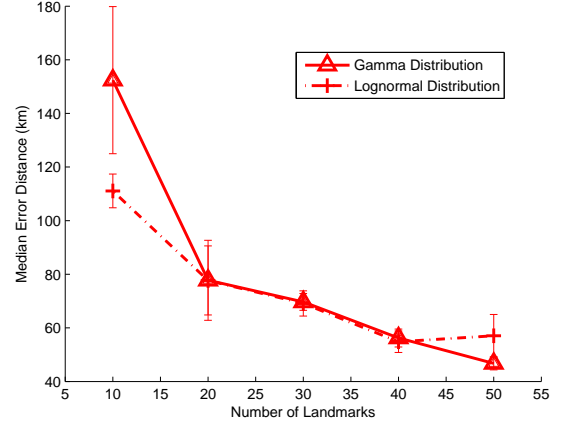


Figure 9: Median geolocation error as a function of the number of landmarks for Gamma and Lognormal noise distributions with  $m = 4$  and  $v = 16$ .

## 5.2 Evaluation on Real Data

This section evaluates GeoWeight by geolocating real targets on the Internet.

### 5.2.1 Experimental Setup

Our experimentation was carried out on PlanetLab ([www.planet-lab.org](http://www.planet-lab.org)), which is an experimental test bed on the Internet. We collected latency data, generated via ping tool, from PlanetLab nodes consisting of 50 landmarks in North America. The location of the landmarks is shown in figure 10. North America covers large geographical area and possesses substantial number of users, hosts and network connectivity of the Internet. Thus, we believe the methodology that we developed in this paper is notable even though we limited our scope to North America. However, the delay-distance relationship may vary based on different geographical location.

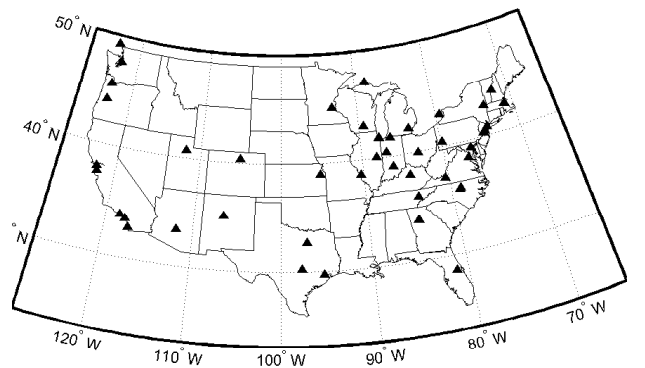


Figure 10: Location of the chosen landmarks

The latency data between landmarks were collected over a period of one month, from September 23, 2008 to October 25, 2008. Over this period, we executed a script, on each landmark, which generated ping commands originating from this landmark to every other landmark. At a given time, the originating landmark performed a three data packet ping to a selected destination landmark followed by a two

minute pause. Then the process continued with a new destination. After cycling through all destination landmarks the process was repeated. From the three observed ping times, the minimum ping time was chosen as the measure of latency for modelling purposes.

The full dataset we gathered consists of approximately 150,000 distance-latency measurements. Although we used 50 landmarks, equal number of measurements were not available for each landmark due to certain ping commands not successfully completing from these PlanetLab nodes during the measurement collection period. The inter-landmark distance in the data covers the range 0.5 km - 4331 km. This data set was used for computing the weights.

During the same period, we also gathered latency data from 50 landmarks as described above to 60 targets in North America. None of our targets was in the same domain as the landmark. This data set was used for geolocating the targets.

### 5.2.2 Experimental Results

We first computed the weights  $w_{ij}$  for the GeoWeight algorithm using the latency measurements between the 50 landmarks. We then geolocated the 60 targets using latency measurements to these targets from the 50 landmarks by choosing one random ping time from each landmark. We geolocated each target four times each time selecting a single random ping time from the data set for each landmark ( $n_i = 1$ ).

Figure 11 shows the cumulative distribution of the mean geolocation error for GeoWeight, Octant and CBG. First of all, since Octant's implementation of the published algorithm was not available to us, we implemented the latency measurement based geolocation step of the Octant algorithm. We did not include the additional optimizations of the Octant algorithm since our goal is to compare with the measurement-only geolocation approach. The maximum and minimum distance bounds in this case were computed, per landmark basis, using the latency data gathered between PlanetLab landmarks. We geolocated the same 60 targets with GeoWeight and Octant using our latency measurements. Figure 11 shows the cumulative geolocation error distribution of GeoWeight and Octant (our implementation) in solid and dotted line respectively. The median geolocation error for GeoWeight is 44 km and for Octant is 456 km.

Octant error is significantly higher compared to the published results in [23]. This we believe is due to two reasons. First, Octant approach is based on minimum and a maximum distance bounds for a given latency. However, with both the PlanetLab (figure 1(a)) and iPlane (figure 1(b)) dataset we have not seen a clear minimum distance bound of distance-latency relationship, even for per landmark basis. The lack of a minimum distance bound is further confirmed in [7] based on their dataset. The lack of a minimum distance bound results in a larger constraint region, resulting in higher error. Second, in the published work [23] Octant used different optimization techniques. Since our goal is to compare the geolocation approaches based purely on latency measurements we did not include such optimization techniques in either algorithm. This gives a fair comparison between the approaches. However, the additional optimizations used by Octant can also be incorporated into our algorithm.

We also geolocated the same 60 targets by Octant and CBG using the geolocation service provided by the authors of Octant [23]. This service is available at: <http://www.cs.cornell.edu/~bwong/octant/query.html>. Median geolocation errors of Octant and CBG by the service provided by the authors of

[23] are 216 km and 506 km respectively. Figure 11 shows the cumulative geolocation error distribution of Octant and CBG obtained from this service in dash and dash-dotted line respectively. We geolocated each target three times and the geolocation error shown in figure 11 is the minimum of the three values. This geolocation service uses supplementary constraints such as geographical and demographical hints in addition to latency measurements to refine its estimates whereas our implementation of Octant does not use any such optimization technique. We believe that this is the reason for the observed differences of geolocation error between our implementation of Octant and the implementation provided by the authors. Extra geographical and demographical hints significantly improved Octant's accuracy and this observation is aligned with the description in [23]. The median and mean estimation errors for the three approaches are listed in table 3.

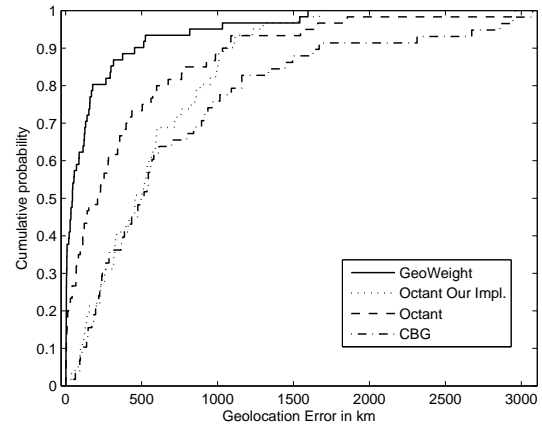


Figure 11: Cumulative distribution of geolocation error distance for GeoWeight, Octant and CBG

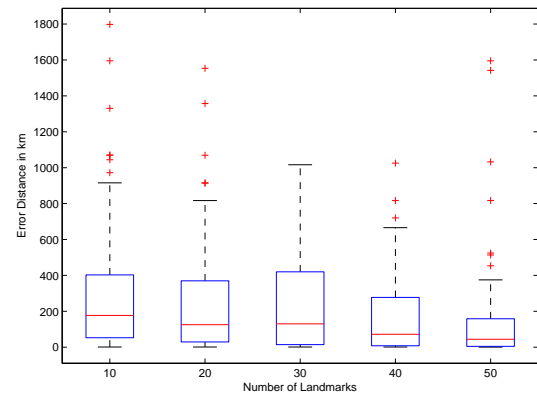


Figure 12: GeoWeight's performance with real data with different number of landmarks

We then investigated the impact of number of landmarks on the performance of GeoWeight. This experiment used a randomly chosen subset of landmarks which were used to geolocate the 60 targets. Figure 12 shows the box plot of geolocation error as a function of the number of landmarks. Figure 12 shows the accuracy increases with increase in the number of landmarks. However, it is to be noted that even with a smaller number of landmarks GeoWeight

Table 3: The mean and median geolocation error (rounded to the nearest kilometer) for GeoWeight, Octant and CBG approaches

	GeoWeight	Octant Our Imp.	Octant	CBG
Mean (km)	170	541	396	749
Median (km)	44	456	216	506

still performs better than existing approaches (Table 3).

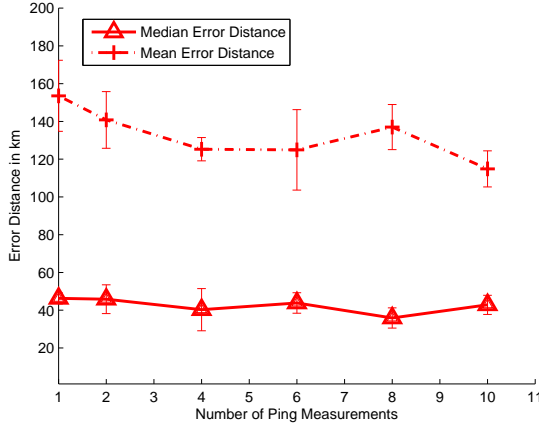


Figure 13: GeoWeight's Performance with multiple ping times

Finally, we investigated the impact of number of measurements,  $n_i$ , between each landmark and target pair on the geolocalization error of GeoWeight.

Figure 13 shows GeoWeight's performance for different values of  $n_i$ . The mean and median geolocation error for 60 targets using 50 landmarks is shown in figure 13. The experiment was repeated four times and the error-bars display the standard deviation of these four results.

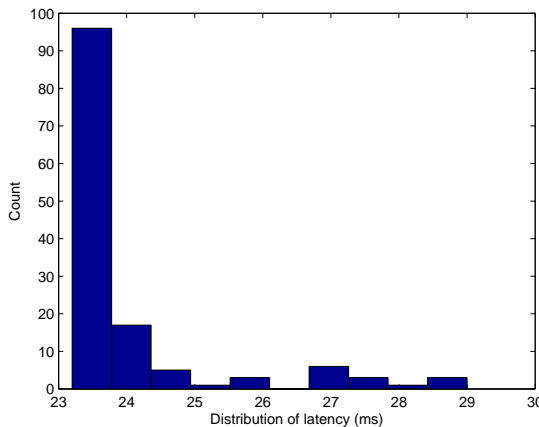


Figure 14: Distribution of latency between Boston University and University of Chicago

It is observed that there is no significant difference in the observed error based on  $n_i$ . In our dataset, we have observed that for a given landmark-target pair most of the ping times observed are similar with some outliers as shown in the histogram in figure 14, which could explain why  $n_i$  does not have an impact on the geolocation error.

## 6 Conclusions

In this paper, we have presented GeoWeight, a novel latency-based geolocation algorithm. GeoWeight is based on a probability model for Internet latency computed from observed latencies for different distances. We validated GeoWeight with simulated and real Internet data using PlanetLab as a test bed. The results show that GeoWeight outperforms existing approaches such as CBG and Octant. GeoWeight was able to geolocate 60 targets in North America using 50 landmarks with a median geolocation error of approximately 44 km. This geolocation approach was based purely on latency measurements as opposed to the existing techniques that supplement the latency data with geographical constraints in order to increase their accuracy.

As our future work we are investigating better probability distributions for Internet latency data which can capture the behavior of noise in latency on a per landmark basis. One initial investigations have focussed on the Levy distribution which have shown promising results.

## References

- [1] iPlane: Latency Measurements from University of Washington. <http://www.mcs.anl.gov/olson/IPtoLL.html>.
- [2] IP to Latitude/Longitude. <http://www.mcs.anl.gov/olson/IPtoLL.html>.
- [3] B.A.FOROUZAN. *Data Communications And Networking*. Tata McGraw-Hill Publishing Company Limited, 2000.
- [4] BASSET, E., JOHN, P., ANDERSON, T., KRISNAMURTHY, A., CHAWATHE, Y., AND WETHERALL, D. Towards IP Gelocation Using Delay and Topology Measurements. *In the Proceedings of IMC 06*. (2006).
- [5] C.GUO, Y.LIU, W. H. Q. Y. Mining the Web and the Internet for Accurate IP Address Geolocations. *In Proceedings of INFOCOM 2009. The 28th Conference on Computer Communications. IEEE* (2009).
- [6] DABEK, F., COX, R., KAASHOEK, F., AND MORRIS, R. Vivaldi: A Decentralized Network Coordinate System. *In the proceedings of the SIGCOMM 2004* (2004).
- [7] D.LI, J.CHEN, C. Y. J. Z. Y. IP-Geolocation Mapping for Involving Moderately-Connected Internet Regions. *Project participation from Microsoft Research: <http://research.microsoft.com/en-us/people/danil/>* (2009).
- [8] E.LIMPERT, W.A.STAHEL, M. Log-normal Distributions across the Sciences: Keys and CLues. *In the Proceedings of BioScience, Vol. 51 No. 5 pp. 341-352*. (2001).
- [9] F.MELAKESSOU, U.SORGER, Z. On The Road Towards The Comprehension Of The Internet Traffic Behavior: Simulation And Analysis Of An End-To-End Connection With NS-2. *In the Proceedings of the 2007 spring simulaiton multi-conference - Volume 1*. (2007).
- [10] FOSSEN, E., AND ARNES, A. Forensic Geolocation of Internet Addresses using Network Measurements. *Nordsec 2005, 10th Nordic Workshop on Secure IT-systems* (2005).

- [11] FRANCIS, P., JAMIN, S., JIN, C., JIN, Y., RAZ, D., SHAVITT, Y., AND ZHANG, L. IDMaps: a global Internet host distance estimation service. *IEEE/ACM Transactions on Networking, Volume 9, Issue 5, Oct. 2001* Page(s):525 - 540 (2001).
- [12] GUEYE, B., UHLIG, S., ZIVIANI, A., AND FDIDA, S. *Networking 2006. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems*. Springer Berlin / Heidelberg, 2006, ch. Leveraging Buffering Delay Estimation for Geolocation of Internet Hosts, pp. 319–330.
- [13] GUEYE, B., ZIVIANI, A., CROVELLA, M., AND FDIDA, S. Constraint-Based Geolocation of Internet Hosts. *Networking, IEEE/ACM Transactions 14*, 6 (2006), 1219–1232.
- [14] GUYTON, J. D., AND SCHWARTZ, M. F. Locating nearby copies of replicated Internet Servers. *In the Proceedings of ACM SIGCOMM 95, Cambridge, MA. U.S.A.* (1995).
- [15] MADHYATHA, H. V., ANDERSON, T., KRISHNAMURTHY, A., SPRING, N., AND VENKATARAMANI, A. A Structural Approach to Latency Prediction. *In the Proceedings of IMC 06, Rio de Janeiro, Brazil* (2006).
- [16] MOORE, D., PERIAKARUPPAN, R., AND DONOHOE, J. Where in the World is netgeo.caida.org?. *Presented in INET2000 Poster (Yokohama, Japan, July 2000)* (2000).
- [17] MUIR, J., AND OORSCHOT, P. C. Internet Geolocation and Evasion. *Technical Report TR-06-05, School of Computer Science, Carleton University*. (2006).
- [18] PADMANABHAN, V. N., AND SUBRAMANIAN, L. An investigation of geographic mapping techniques for internet hosts. *In the Proceedings of ACM SIGCOMM Computer Communication Review* (2001).
- [19] PIETZUCH, P., LEDLIE, J., MITZENMACHER, M., AND SELTZER, M. Network-Aware Overlays with Network Coordinates. *In the Proceedings of 26th IEEE International Conference on Distributed Computing Systems Workshops, ICD-CSW06* (2006).
- [20] ROXIN, A.; GABER, J. W. M. N.-S.-M. A. Survey of Wireless Geolocation Techniques. *In the proceeding of the IEEE Globecom Workshops, 2007*. (2007).
- [21] S, L., N., V., AND H., R. Geographic Properties of Internet Routing. *Proceedings of the General Track: 2002 USENIX Annual Technical Conference* (2002).
- [22] WANG, B., SLIVKINS, A., AND SIRER, E. M. Meridian: A Lightweight Network Location Service without Virtual Coordinates. *In the Proceedings of ACM SIGCOMM 05, Philadelphia, Pennsylvania, U.S.A.* (2005).
- [23] WONG, B., STOYANOV, I., AND SIRER, E. Octant: A Comprehensive Framework for the Geolocalization of Internet Hosts. *In Proceedings of Symposium on Networked System Design and Implementation, Cambridge, Massachusetts* (2007).
- [24] ZIVIANI, A., FDIDA, S., DE REZENDE, J. F., AND DUARTE, O. C. M. B. Improving the accuracy of measurement-based geographic location of internet hosts. *Computer Networks and ISDN Systems archive, Volume 47 , Issue 4 (March 2005)* (2005).



# Joined Q-ary Tree Anti-Collision for Massive Tag Movement Distribution

Prapassara Pupunwiwat

Bela Stantic

Institute for Integrated and Intelligent Systems  
Griffith University, Queensland, Australia  
Email: {p.pupunwiwat, b.stantic}@griffith.edu.au

## Abstract

Radio-Frequency Identification (RFID) systems consist of tags and networked electromagnetic readers. Despite the emergence of RFID technology, the problem of identifying multiple tags, due to the *Collisions* is still a major problem. The problem can be solved by using anti-collision methods such as *ALOHA-based* approaches and *Tree-based* approaches. ALOHA-based approaches suffer from tag starvation, which causes that not all tags can be identified. The tree-based approaches suffer from too long identification delay caused by lengthy queries during identification process. In this paper, we propose a tree-based anti-collision method called “Joined Q-ary Tree”, which adaptively adjusts tree branches according to tag movement behavior and number of tags within an interrogation zone. In this empirical study, we demonstrate that the proposed method is suitable for numerous scenarios. It requires less queries issued per complete identification than existing approaches while ensuring identification of all tags within the interrogation zone.

**Keywords:** Radio Frequency Identification - RFID, Anti-Collision

## 1 Introduction

RFID technology uses radio-frequency waves to automatically identify people or objects. Currently, RFID technology is used in different systems such as: transportation, distribution, retail and consumer packaging, security and access control, monitoring and sensing, library system, defence and military, health care, and baggage and passenger tracing at the airports. RFID reader retrieves information from tags and sends that information back to host computer via middleware. RFID data, which is captured by readers, can be accumulated very fast and does not carry much information as it is raw. These data are inaccurate and they need to be filtered in order to improve database management.

The main issue that usually occurs in RFID data streams is data errors. The most common errors are Missed reads, which are caused by collisions where Radio Frequency (RF) signals interfere with each other, preventing the reader from identifying tags. RF collision problem can be solved by using anti-collision techniques to prevent two or

more tags from responding to a reader at the same time. Tag anti-collision algorithms can be categorised into ALOHA-based algorithms and Tree-based algorithms. ALOHA-based algorithms suffer from Tag starvation problems such that not all tags can be identified. On the other hand, Tree-based algorithms make trees while performing the tag identification procedure using a unique ID of each tag, which lead to lengthy queries issued by reader that causes long identification delay.

In this paper, we present a “Joined Q-ary Tree”, which is based on Memoryless Q-ary Tree. We performed extensive experimental study in order to prove the efficiency of the proposed technique. The results and analysis of the experiments indicate that “Joined Q-ary Tree” can more effectively reduce queries length and improve the system efficiency than the Naive Q-ary Trees.

The remainder of this paper is organised as follows: In Section 2, some general background are provided on RFID and information related to Tag collision and Massive Tag Movement. In Section 3, we discuss related works on our proposed method and their limitations, which include ALOHA-based and Tree-based anti-collision, and Query tree protocols. In Section 4, we present a new technique, the Joined Q-ary Tree including methodology and scenarios. In Section 5, we present experimental evaluation, results, analysis and discussions; and finally in Section 6 we provide our conclusion and future work.

## 2 RFID Background

RFID technology is an automatic identification technology of contactless method that identifies electronic tags attached to items. There are several methods of identification but the most common is to store a serial number that identifies a person or object such as Electronic Product Code (EPC). RFID may only consist of a tag and a reader but a complete RFID system involves many other technologies, such as computer, network, Internet, and software such as middleware and user applications.

### 2.1 Electronic Product Code (EPC)

EPC Class 1 or passive EPC tag is widely used in Ultra High Frequency (UHF) range for communications at 860-960MHz, where the standards have been created by EPCGlobal (EPCGlobal 2006). The most common encoding scheme currently widely used includes: General Identifier (GID-96), Serialised Global Trade Item Number (SGTIN-96), Serialised Shipping Container Code (SSCC-96), Serialised Global Location Number (SGLN-96), Global Returnable Asset Identifier (GRAI-96), Global Individual Asset Identifier (GIAI-96), and DoD Identifier (DoD-96).

In this paper, we only focus on **General Identifier 96-bits** type to evaluate our approach. The implementation and experiment will be determined by the impact of *EPC Pattern* and *Massive tag movement*. Therefore at present, only one type of encoding is necessary.

GID-96	Bit	Max.Decimal/Binary
<b>Header</b>	8	0011 0101
<b>GMN*</b>	28	268,435,455
<b>Object Class</b>	24	16,777,215
<b>Serial Number</b>	36	68,719,476,735

Table 1: The General Identifier (GID-96) includes three fields in addition to the *Header* with a total of 96-bits binary value (\*GMN = General Manager Number).

The general structure of EPC tag encodings is a string of bits, consisting of a fixed length (8-bits) *Header* followed by a series of numeric fields whose overall length, structure, and function is completely determined by the *Header* value. Table 1 shows an example of GID-96 EPC passive tag encoding scheme. Only *Header* is shown in binary; the rest are shown in decimal number.

## 2.2 Massive RFID Tags Movement

Items tend to move and stay together through different locations especially in a large warehouse (Gonzalez, Han & Li 2006), (Gonzalez, Han, Li & Klabjan 2006). For example, 4 pallets - with 24 cases of crystal wine glasses, plates, bowls, and jugs - each may be ready to leave the warehouse and deploy to different retailer. At this point, 4 pallets move along a conveyor belt through dock doors mounted with RFID readers. We can, for example, use the assumption that many RFID objects stay or move together, especially at the early stage of distribution. These EPC data will be very similar since the first few bits of encoding will determine the type of *Encoding Scheme* (Header), *Company Prefixes* (GMN) and *Object Class*.

Currently, for example in warehouse distribution environment where RFID systems are deployed, the UHF range of RF waves and passive tags are used for long distance identification. In order to manage and monitor the traffic of RFID data effectively, *EPC pattern* is usually used to keep unique ID on each item arranged within a specific range. *EPC pattern* does not represent a single tag encoding, but rather refers to a set of tag encodings. For example, there are 20 cases of wine-glasses packed together in the same pallet. *EPC pattern* ranging between 1 to 20 may be used so that a foreign tag, e.g. an EPC tag with ID 47, can be filtered out easily if accidentally captured by the reader.

## 2.3 RFID Data Stream Errors

Due to the low-power and low-cost constraints of RFID tags, reliability of RFID readings is of concern in many circumstances (Brusey et al. 2003), (Vogt 2002). There are four typical errors: *Unreliable reads*, *Noise*, *Missed reads*, and *Duplication*. Several techniques for filtering RFID data have been proposed in literatures (Bai et al. 2006), (Jeffery et al. 2005), (Jeffery, Garofalakis & Franklin 2006), (Jeffery, Alonso, Franklin, Hong & Widom 2006), (Carbunar et al. 2005), (Fishkin et al. 2004). However, these techniques only filter specific kind of errors. A research

on *Noise* and *Duplication* data filtering has been done very well previously; nonetheless, the *Unreliable reads* can be prevented only at some point. This depends on the deployment of readers, tags, and an environment.

Missed Reads are very common in RFID applications, which are caused by collisions of RF signals that interfere with each other preventing the reader from identifying tags. RF collisions can be solved by performing anti-collision at the edge to prevent two or more tags from responding to a reader at the same time. RF collision can happen at two levels: Collision at reader level and Collision at tag level.

## 2.4 Tag and Reader Collision

Simultaneous transmissions in RFID systems lead to collisions as the readers and tags typically operate on the same channel. Three types of collisions are possible: Reader-Reader collision, Reader-Tag collision, and Tag-Tag collision.

- **Reader-to-Reader:** Interference occurs when one reader transmits a signal that interferes with the operation of another reader; and prevents the second reader from communicating with tags in its interrogation zone (Jain & Das 2006). Reader-to-Reader collision can be easily avoided by determining the appropriate reader's deployment that prevents direct signal interference between two or more readers.
- **Reader-to-Tag:** Interference occurs when one tag is simultaneously located in the interrogation zone of two or more readers, where more than one reader attempts to communicate with that tag at the same time (Jain & Das 2006).
- **Tag-to-Tag:** Tag collision in RFID systems, also known as Multi-Access, happens when multiple tags are energised by the RFID tag reader simultaneously, and reflect their respective signals back to the reader at the same time. This problem is often seen whenever a large volume of tags must be read together in the same reader zone because the reader is unable to differentiate these signals.

## 3 Related Works

Tag collision or Multi-Access problem is more complex than those within Reader collision categories. Therefore, in this paper we only focus on tag anti-collision approaches.

### 3.1 Tag Anti-Collision Approaches

The various types of tag anti-collision methods for Multi-access/Tag collision can be reduced to two basic types: ALOHA-based method and Tree-based method. In an ALOHA-based method, tags respond at randomly generated times. If a collision occurs, colliding tags will have to identify themselves again after waiting a random period of time. This technique is faster than Tree-based but suffers from Tag starvation problem where not all tags can be identified due to the random nature of chosen time.

The Tree-based method starts by asking for the first number of the tag (Query Tree algorithm) until it matches the tags; then it continues to ask for additional characters until all tags within the region are found. This method is slow and introduces a long Identification delay but leads to fewer collisions, and have 100 percent successful identification rate.



### 3.1.1 ALOHA-Based Methods

The ALOHA-based methods usually refers to “Slot ALOHA” (Quan et al. 2006), which introduces discrete time-slots for tags to be identified by reader at the specific time. The principle of Slotted-ALOHA techniques is based on the “Pure ALOHA” introduced in early 1970s (Abramson 1970), where each tag is identified randomly. To improve the performance and throughput rate, a “Frame Slotted ALOHA” (Shin et al. 2007) anti-collision algorithm has been proposed, where each frame is formed of specific number of slots that is used for communication between the readers and the tags. Each tag in the interrogation zone arbitrarily selects a slot for transmitting the tag’s information. Several researchers (Wang et al. 2007), (Lee et al. 2005), (Lee et al. 2008), (Cho et al. 2007) have attempted to improve the throughput rates by implementing a Frame Estimation Tool. However, the ALOHA-based method can only be improved to a very high throughput rate but they still cannot achieve a hundred percent tag identification.

### 3.1.2 Tree-Based Methods

Such Tree-based methods can be classified into a Memory-based algorithm and a Memoryless-based algorithm. In the Memory-based algorithm, which can be grouped into a splitting tree algorithm such as an “Adaptive Splitting Algorithm” and a “Bit Arbitration Algorithm,” the reader’s inquiries and the responses of the tags are stored and managed in the tag memory, resulting in an equipment cost increase especially for RFID tags. In contrast for the Memoryless-based algorithm, the responses of the tags are not determined by the reader’s previous inquiries. The tags’ responses and the reader’s present inquiries are determined only by the present reader’s inquiries so that the cost for the tags can be minimised. Memoryless-based algorithms include a “Query Tree Algorithm”, a “Collision Tracking Tree Algorithm”, and a “Tree Walking Algorithm.”

In this paper, we will focus on Memoryless Query Tree based protocols since it is the most popular and is effective anti-collision technique for passive UHF tags. However, there are other improved anti-collision methods based on Query Tree such as an “Adaptive Query Splitting” (AQS) proposed by Myung & Lee (2006b), Myung & Lee (2006a); and a “Hybrid Query Tree” (HQT) proposed by Ryu et al. (2007). AQS keeps information, which is acquired during the last identification process in order to shorten the collision period. This technique requires tags to support both the transmission and reception at the same time, thereby making it difficult to apply to low-cost passive RFID systems. HQT uses a 4-ary query tree instead of a binary query tree, which increased too many *Idle cycles* despite reducing *Collision cycles*; while extra memory needed also increases as an identification process gets longer, because each query increases the prefixes by 2-bits instead of 1-bit. Accordingly, the Query Tree algorithm, adopted at present as the anti-collision protocol in EPC Class 1, may be limited to the tree based anti-collision protocol, which can be implemented (Choi et al. 2008).

## 3.2 Query Tree Based Protocols

The Query tree is a data structure for representing prefixes which is sent by the reader in Query tree protocols. A reader identifies tags through an uninterrupted communication with tags. The Query tree protocols consist of loops, and in each loop, the reader transmits a query with specific prefixes, and the tags respond with their IDs. Only tags with IDs

that match the prefixes, respond. When only one tag responds to reader, the reader successfully recognises the tag. When more than one tag tries to respond to reader’s query, tag collision occurs and the reader cannot get any information about the tags. The reader, however, can recognise the existence of tags to have ID that matches the query. For identifying tags that lead to the collision, the reader tries to query with 1-bit longer prefixes in next loops. By extending the prefixes, the reader can recognise all the tags.

Depending on the number of tags that respond to the interrogator, there are three cycles of communication between tag and reader.

- **Collision cycle:** Number of tags that respond to the reader is more than one. The reader cannot identify the ID of tags.
- **Idle cycle:** No response from any tag. It is a waste that should be reduced.
- **Success cycle:** Exactly one tag responds to the reader. The reader can identify the ID of the tag.

The delay in identification of tags is mostly affected by the *Collision cycles*, *Idle cycles*, and *Massive Tag Movement*. Therefore, minimising the number of *Collision cycles* and eliminating *Idle cycles* can improve the identification ability of the reader because *Bits Length* required by the reader can be reduced.

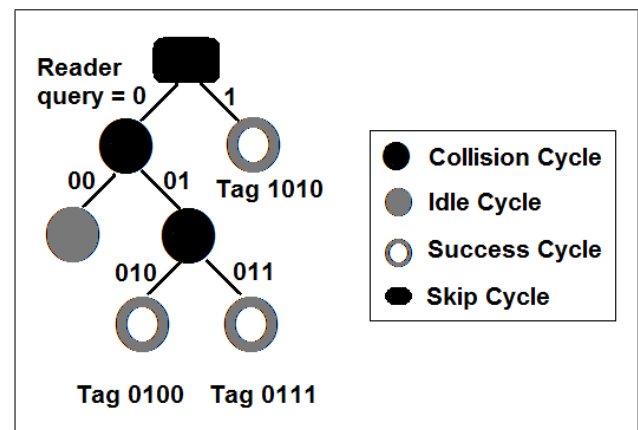


Figure 1: Tree-based anti-collision protocols: Memoryless Query Tree.

Figure 1 displays an example of a QT procedure. An identification process starts at Level one of tree, where QT uses tag IDs to split a tag set. It can be seen that *Tag 1010* is successfully identified in the first round because from all three tags, only *Tag 1010* has ‘1’ for the first bit of string. In the second round of identification, *Idle cycle* was created as there was no tag starting with ‘00’ for the first two bits. In the third round of identification, the other two tags, *Tag 0100* and *Tag 0111*, are successfully identified.

To overcome shortcomings of existing methods for anti-collision, we propose a “Joined Q-ary Tree” based on Memoryless Q-ary Tree, which adaptively adjust their tree branches according to Number of tags. We focus on analysing the impact of *Massive Tag Movement* within warehouse, therefore, we only considered static tags where tags have no mobility.

## 4 Methodology

In order to minimise the length of queries issued by a reader, a “Joined Q-ary Tree” or a combination of

two Q-ary trees is proposed. The Joined Q-ary Tree is based on a Memoryless Q-ary Tree anti-collision protocol, where no other memory is required beside a tag's IDs. This section will describe the scenarios where massive EPC tag movement exists and *EPC pattern* are used; a classification of *Splitting Fitness*; the *Joined Q-ary Tree* with Sample Tag Splitting; and the Tags Prediction and Classification for *Unique Bits* of EPC.

### 4.1 Warehouse Distribution Scenarios

In this work, a specific scenario is examined based on *Massive tag movement*. The paper focus's is on Crystal warehouse scenario, which can be classified into four different scenarios, as follows:

- **Scenario One:** All items have the same Encoding Schemes, same Company Prefixes, same Object Class, but different Serial Number (Figure 2a).
- **Scenario Two:** All items have the same Encoding Schemes and same Company Prefixes; but different Object Class and different Serial Number (Figure 2b).
- **Scenario Three:** All items have the same Encoding Schemes; but different Company Prefixes, different Object Class, and different Serial Number (Figure 2c).
- **Scenario four:** All items have different Encoding Schemes, different Company Prefixes, different Object Class, and different Serial Number (Figure 2d).

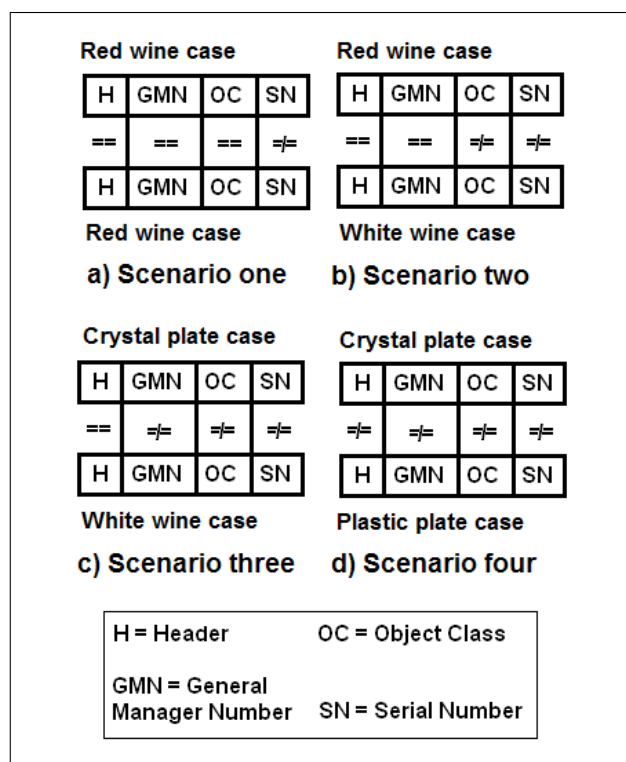


Figure 2: Crystal Warehouse Scenario: a) Two cases of Red-wine, b) White-wine case and Red-wine case, c) White-wine case and Crystal plate case, and d) Crystal plate case and Plastic plate case.

Out of the four scenarios, Scenario one and Scenario two will most likely occur in a large warehouse

distribution. For example, there are 5 pallets of: Red-wine glasses; White-wine glasses; Champagne glasses; Beer glasses; and Tumbler; with 12 cases each that are ready to leave the warehouse to a smaller retailer. If all of 5 pallets move into an interrogation zone at the same time, we will have the data with five different *Object Class* (OC) and sixty different *Serial Numbers* (SN). Nevertheless, since all items are from the same company that produced crystal-ware, they will all have the same *Header* and *General Manager Number*. When tags from the same pallet collide, we will have two or more tags with the same *Header*, same *General Manager Number*, same *Object Class*, and unique *Serial Number* which are related to Scenario one. On the other hand, when tags from different pallets collide, we will have two or more tags with the same *Header*, same *General Manager Number*, different *Object Class*, and different *Serial Number*, which are related to Scenario two.

In addition, a data management in a large warehouse distribution can be improved by using *EPC pattern*. As explained earlier, *EPC pattern* is referred to as a set of tag encodings. A sample of *EPC pattern* is: 25.1545.[3456-3478].[778-795] in decimal, which later will be encoded to binary and embedded onto tags. Thus, within this sample pattern, *Header* is fixed to 25 and *Company Prefixes* is 1545; while *Object Class* can be any number between 3456 and 3478; and *Serial Number* can be anything between 778 and 795.

In this paper, we will focus on data captured according to Scenario one and Scenario two, and *EPC pattern* will be used to create different set of tag encodings.

## 4.2 Splitting Fitness

*Splitting fitness* can be classified into *Worst-case splitting*, *Perfect splitting*, and *Random splitting*. All three cases are discussed below:

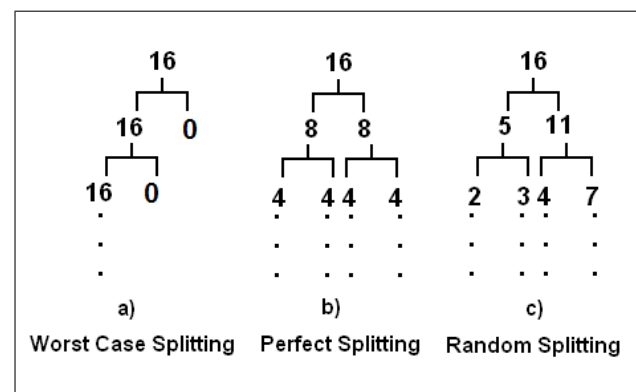


Figure 3: Splitting Fitness: a) Worst-Case Splitting, b) Perfect Splitting, and c) Random Splitting.

### 4.2.1 Worst-Case Splitting

*Worst-Case splitting* is when tags spliced into an unbalanced tree, where one child node has no further node in a binary tree case. Figure 3a) shows that there are 16 tags at Level 0 tree; then at Level 1, tags spliced into 16 tag on the left-hand node and no tag on the right-hand node. No further splitting is necessary on the right-hand node, since there is no tag left. This case of splitting will likely happen for the first few bits of EPC identification in real world warehouse environment because most items have *Massive tag movement* and usually belong to the same *EPC pattern* with

similar ID. The worst case splitting caused more *Idle cycles* especially for higher level trees (4-ary, 8-ary, and 16-ary) because all tags will be travelling down to only one side of the tree, which results in further collision.

#### 4.2.2 Perfect Splitting

*Perfect splitting* happens when a set of tags spliced to the left and right child node equally. Figure 3b) shows that there are 16 tags at Level 0 tree; then at Level 1, tags spliced equally into 8 nodes. Further splitting is required for both left-hand and right-hand nodes until only one tag is left. This case of splitting is impossible in real world scenario but will be the closest case to the latter stages (bits) of EPC identification within warehouse environment because most items belong to the same group of *EPC pattern*. For example, 1 pallet of White-wine glasses containing 20 cases move into one interrogation zone. All items from the pallet will have the same *Object Class* and will travel along the same side of child node at earlier Levels; resulting in *Worst Case Splitting*. However, the remaining few bits will be unique for each EPC since they belong to *Serial Number*. These remaining bits encoded within the same *EPC pattern* will split almost equally to the left and right child nodes. Both child nodes of left-hand side and right-hand side of binary tree will not be exactly equal since data captured are not always even. Therefore, we can call this situation as *Almost-Perfect Splitting*.

#### 4.2.3 Random Splitting

*Random splitting* happens when a set of tags spliced to the left and right child node randomly and splitting pattern cannot be found. Figure 3c) shows that there are 16 tags at Level 0 tree; then at Level 1, tags spliced into 5 and 11. Further at Level 2, tags spliced into 2, 3, 4, and 7 where no specific splitting pattern exists; thus, this situation is called *Random Splitting*. This case of splitting will likely happen in retail distribution environments since all items usually come from different locations. Thus, this splitting case will not be further discussed as this research will only focus on warehouse environment.

### 4.3 Joined Q-ary Tree

Query Tree uses each bit of tag ID to split a tag set. Q-ary tree uses every 2-, 3-, or 4-bits of tag ID to split a tag set. Q-ary tree increases the child node of tree from '2' to '4', '8' or '16' nodes and so on. This way, we can reduce collisions but at the same time, the *Idle cycles* will increase, along with the *Bits Length* needed in each query. The best way to solve this problem is to employ the right combination of Q-ary trees for each specific scenario. This will depend on the specific Number of tags within an interrogation zone, *Massive Tag Movement*, and *Splitting Fitness* based on *EPC pattern*.

The Joined approach is a combined Q-ary trees, specifically 2-ary Tree and 4-ary Tree, which have been identified to be the best Q-ary trees in (Pupun-wiat & Stantic 2009). The Joined approach will be applied on each collided tags EPC, which will be split using every 1 or 2-bits of tag ID for the first few queries; and then at one point, every 1 or 2-bits will be queried. With the fact that most items from warehouse have massive movement, first few bits of EPC will be identical and the remaining bits will be very similar. In order to optimise the performance of Joined Q-ary Tree, the right Separating Point (SP) between the two Q-ary trees needs to be configured.

The objective of Joined Q-ary Tree is to reduce the *Bits Length* queried by a reader so that Identification time can be minimised. In this paper, we will investigate and compare the "Naive Q-ary Tree" approach and our newly proposed "Joined Q-ary Tree".

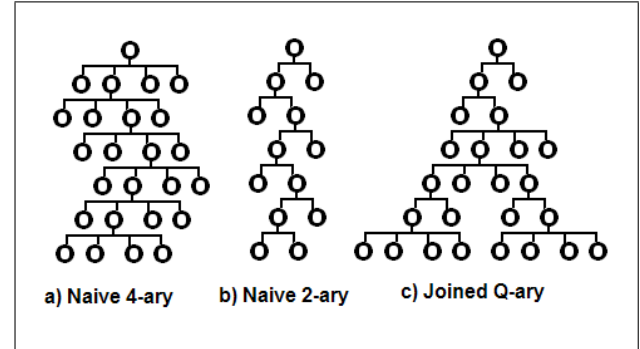


Figure 4: A sample of: a) a Naive 4-ary Tree, b) a Naive 2-ary Tree, and c) a Joined Q-ary Tree.

Figure 4 shows the example of a) Naive 2-ary, b) Naive 4-ary, and c) Joined Q-ary Tree. Joined Q-ary Tree bonded both 2-ary and 4-ary tree together and apply to specific bits of EPC depending on how Identical or Unique they are. The classification of *Identical bits* and *Unique bits* will be explained further within this section.

#### 4.3.1 EPC Bits Prediction and Classification

In warehouse distribution environment according to Scenario one and Scenario two, it is known that the first 36-bits of EPC (Header and GMN) are definitely Identical. However, 24-bits of *Object Class* can be both Identical and Unique for all tags, depending on how many pallets existed within one interrogation zone. For example, if there are 5 pallets of 12 cases each in the interrogation zone, there will be five different *Object Class* and sixty unique *Serial Numbers* for all sixty items (cases).

Since *Object Class* involved 24-bits of EPC (allow 16,777,215 unique tags) but only 5 unique OC is needed, we must calculate a certain number of *Unique bits* needed in order to apply the right Q-ary tree. This also applied to *Serial Number* that contains 36-bits of string. Assuming that *EPC pattern* is used, not all 36-bits of these strings will be Unique.

Table 2 shows a formal structure for bits classification of GID-96 bits EPC. It can be seen that the *Identical bits* of EPC always equal to 36-bits for the first 36-bits of EPC. This includes 8-bits of *Header* and 28-bits of *GMN*, which are always the same for all tags. For *Object Class*, 24-bits are available where *Unique bits* within *Object Class* (UOC) can be predicted using Equation (1). In addition, *Serial Number* with 36-bits can also be predicted using the same Equation.

	Length	Identical	Unique
<b>Header</b>	8	8	0
<b>GMN</b>	28	28	0
<b>Object Class</b>	24	24 - UOC	UOC*
<b>Serial Number</b>	36	36 - USN	USN**

Table 2: Formal structure of bits classification of EPC GID-96 bits. \*UOC is number of Unique bits within *Object Class* and \*\*USN is number of Unique bits within *Serial Number*.

Our method is executed based on the assumption that the approximate Number of tags (pallets, cases)



is known prior to the identification process. This information is needed for *Unique bits* calculation: UOC, and USN from table 2. However, in most circumstances, Number of tags is usually unknown until the first query issued by the reader. Therefore, UOC and USN of Joined Q-ary Tree can be initially set to zero and after the first round of identification, these two parameters can be calculated.

Joined Q-ary Tree adaptively adjusts their tree branches at specific Separating Point. These SP is configured according to *Identical bits* and *Unique bits* within an EPC data. In order to calculate the estimated number of *Unique bits* within an EPC, we need the average Number of tags within an interrogation zone, and then to apply the equation below:

$$B = \log_2(N) \quad (1)$$

Where N = Number of tags, B = *Unique Bits* of EPC.

Figure 5 illustrates tag splitting behaviour of massive tag within a warehouse. The first 36-bits of EPC belongs to *Header* and *GMN*, therefore 2-ary Tree is applied to these bits since it is simplified to be the most suitable tree for *Identical bits* of EPC. UOC and USN on the other hand, can be split using 4-ary Tree since it is proven to be the most suitable tree for *Unique bits* of EPC.

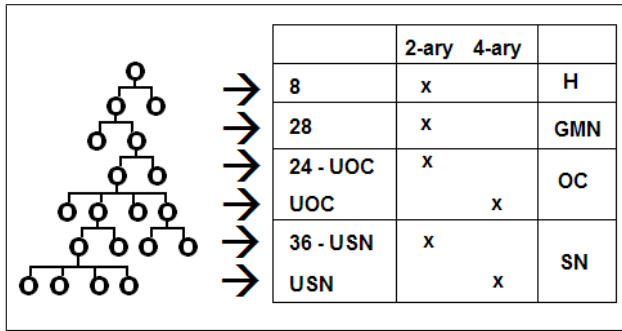


Figure 5: Joined Q-ary Tree structure for GID-96 bits EPC.

#### 4.3.2 Sample Bits Prediction

To demonstrate a calculation of *Unique bits* of EPC, we examine a *Massive tag movement* of 720 tags within 12 pallets (OC). By using Equation (1), UOC and USN can be calculated as follows:

$$UOC = \log_2(N) = \frac{\log_{10}(N)}{\log_{10}(2)} = \frac{\log_{10}(12)}{\log_{10}(2)} \approx 4$$

$$USN = \log_2(N) = \frac{\log_{10}(N)}{\log_{10}(2)} = \frac{\log_{10}(60)}{\log_{10}(2)} \approx 6$$

Therefore, number of *Unique bits* required to cover all unique *Object Class* is approximately 4-bits and approximately 6-bits for *Serial Number*.

Table 3 shows a sample structure for bits classification of 720 tags with GID-96 bits EPC encoding scheme. Corresponding with figure 6, we can see that the *Identical bits* of EPC always equal to 36-bits for the first 36-bits of EPC. 2-ary Tree is applied to these bits. *Identical bits* of *Object Class* (20-bits) and *Serial Number* (30-bits) are also spliced by 2-ary Tree. UOC (4-bits) and USN (6-bits) on the other hand, can be split using 4-ary Tree.

	Length	Identical	Unique
Header	8	8	0
GMN	28	28	0
Object Class	24	20	4
Serial Number	36	30	6

Table 3: Sample bits classification of EPC GID-96 bits, where *Object Class* = 12 and *Serial Number* = 60 (Total of 720 tags).

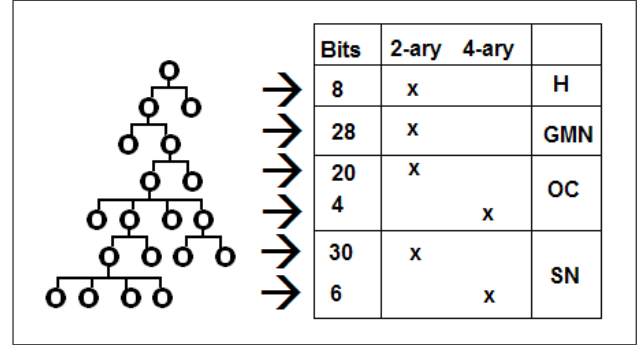


Figure 6: Joined Q-ary Tree structure for GID-96 bits EPC with 36 *Identical bits* Header and GMN, 20 *Identical bits* OC, 4 *Unique bits* OC, 30 *Identical bits* SN, and 6 *Unique bits* SN.

#### 4.3.3 Sample Tags Splitting

We are now initiating a sample comparison between the performance of Naive 2-ary tree, Naive 2-ary, and Joined Q-ary Tree. Table 4 shows 10 sample EPC of 36-bits tags with 24-bits *Identical* and 8-bits *Unique*. We assumed that all *Identical bits* of EPC belong to *Header* and *General Manager Number*, while *Unique bits* of EPC belong to *Object Class* and *Serial Number*.

Identical bits	Unique bits
0011 0101 1010 1101 0000 1010	1011 0101
0011 0101 1010 1101 0000 1010	0100 1011
0011 0101 1010 1101 0000 1010	1101 1100
0011 0101 1010 1101 0000 1010	0010 0100
0011 0101 1010 1101 0000 1010	1100 1000
0011 0101 1010 1101 0000 1010	0001 0100
0011 0101 1010 1101 0000 1010	0011 0011
0011 0101 1010 1101 0000 1010	1110 1010
0011 0101 1010 1101 0000 1010	1111 0010
0011 0101 1010 1101 0000 1010	0000 0110

Table 4: Sample 36 bits tags with 24 bits *Identical* and 8 bits *Unique*.

**Sample Tag Identification** After applying different trees on these tags, we can see that the Naive 2-ary tree has the best performance on *Identical bits* of EPC by using smallest number of queries (*Bits Length*) as shown in table 5. On the other hand, 4-ary tree performed better on *Unique bits* than 2-ary tree.

**Sample Bits Length Calculation** In order to calculate a *Total Bits Length* required for the whole identification process for 2-ary Tree, 4-ary Tree, and Joined Q-ary Tree, information on Number of Child Nodes (NCN) for each level of tree and Number of Bits per Query (NBQ) for that specific level, is

Performances on Identical bits					
Q-ary Tree	Idle Cycles	Collision Cycles	Successful Cycles	Total Cycles	Bits Length
2-ary	24	24	0	48	600
4-ary	36	12	0	48	624
Performances on Unique bits					
Q-ary Tree	Idle Cycles	Collision Cycles	Successful Cycles	Total Cycles	Bits Length
2-ary	0	8	10	18	36
4-ary	2	0	10	12	32

Table 5: Identification process of 2-ary Tree and 4-ary Tree on *Identical bits* and *Unique bits* of EPC.

needed. Number of bits per Level (NBL) can be calculated as follows:

$$NBL = NCN \times NBQ \quad (2)$$

After calculating the NBL for each level of tree, the *Total Bits Length* (NBLs) required can be found by doing the summation of all NBL. After adding all NBL together, the NBLs of 828-bits, 848-bits, and 728-bits are shown in table 6 respectively for 2-ary Tree, 4-ary Tree, and Joined Q-ary Tree.

Level	2-ary	4-ary	Joined
1-12	156	624	156
13-14	54	224	54
15-24	390	0	390
25-26	104	0	128
27-28	124	0	0
<b>NBLs</b>	<b>828</b>	<b>848</b>	<b>728</b>

Table 6: Calculation of *Total Bits Length* required for two Naive Q-ary Trees and a Joined Q-ary Tree. NBLs shows the *Bits Length* required for the specific Naive/Joined Q-ary Tree.

Table 7 shows the overall calculation of *Bits Length* queried on *Identical bit* and *Unique bits* of EPC. We can see that the Joined Q-ary Tree required the least *Total Bits Length* compared to the two Naive Q-ary Trees. Joined Q-ary Tree reduced *Bits Length* by almost 15% compared to Naive 4-ary Tree and by 12% compared to Naive 2-ary Tree.

## 5 Experimental Evaluation

To study the proposed “Joined Q-ary Tree” and compare with the performance of Naive approaches, experiments are performed according to a Crystal warehouse scenario. The experiment is assumed to be set

Different Q-ary Tree on Sample Tags							
Q-ary Tree	Idle Cycles	Collision Cycles	Successful Cycles	Total Cycles	Identical Bits	Unique Bits	Total Bits Length
2-ary	24	32	10	66	600	228	828
4-ary	38	12	10	60	624	224	848
Joined	26	24	10	60	600	128	728

Table 7: Performance Analysis of 2-ary, 4-ary, and Joined Q-ary on set of 10 sample tags.

up in a well controlled environment where there is no metal or water nearby. A UHF RFID reader is used and mounted on a dock door at the end of a conveyor belt. Passive RFID tags are attached to each case of crystal ware. Each pallet of wine glasses, plates, and bowls are moved along this conveyor belt. At this stage, we assume that all three pallets move-in and move-out at the same time to an interrogation zone and no Arriving tag or Leaving tags are present during each identification round.

**Specification:** An Intel Core 2 CPU with 2.40GHz processor and 3GB RAM is used for testing. A Microsoft Window XP professional with Service Pack 3 is installed on the computer. Algorithms for data simulation are implemented using Java JCreator.

### 5.1 Data Set

In this study, we conducted an experiment using three different tag sets: 288 tags, 576 tags, and 864 tags. The impact of different Number of tags in an interrogation zone and performances of difference approaches are to be evaluated. Data sets using *EPC pattern* from table 8 are used:

	EPC Pattern
<b>H</b>	0011 0101
<b>GMN</b>	104,426,055
<b>OC</b>	[ 9,872,273 - 9,872,308]
<b>SN</b>	[ 26,292,755,245 - 26,292,755,268]

Table 8: The *EPC Pattern* is shown in Decimal for GMN, OC, and SN, but will be encoded in Binary on to each tag attached to the item.

- DATA SET ONE: 12 pallets, 24 cases each
- DATA SET TWO: 24 pallets, 24 cases each
- DATA SET THREE: 36 pallets, 24 cases each

#### 5.1.1 Theoretical Bits Prediction

Assuming that the existence of tags are known before identification process, by using Equation (1), UOC and USN of data set one, two, and three can be calculated as follows:

**Data Set one:**

$$UOC = \log_2(12) \approx 4, USN = \log_2(24) \approx 5$$

**Data Set two:**

$$UOC = \log_2(24) \approx 5, USN = \log_2(24) \approx 5$$

**Data Set three:**

$$UOC = \log_2(36) \approx 6, USN = \log_2(24) \approx 5$$

Therefore, number of *Unique bits* required to cover all unique *Object Class* and *Serial Number* are as shown in table 9.

	288 tags		576 tags		864 tags	
	I	U	I	U	I	U
<b>H</b>	8	0	8	0	8	0
<b>GMN</b>	28	0	28	0	28	0
<b>OC</b>	20	4	19	5	18	6
<b>SN</b>	31	5	31	5	31	5

Table 9: Identical and Unique Bits classification of EPC GID-96 bits for Data set one, two and three. I = Identical bits, U = Unique bits.

### 5.1.2 Actual Separating Point Configuration

After theoretical bits estimation, we assumed that the actual encoding of *Unique bits* may be 1 to 2-bits longer than predicted. Therefore, we added 2-bits to the predicted *Unique bits* of each data set. If the predicted bits added up as odd number, 1 more bit is further attached. For example, UOC of data set two (576 tags) is predicted to be 5-bits long. By affixing additional 2-bits, total of 7-bits are applied to UOC. However, this added up as odd number, therefore, 1 more bit is further added (Total 8-bits). Table 10 shows an actual SP for each data set. At specific SP, the Joined Q-ary Tree will adaptively change its branch to 2-ary or 4-ary. Since 2-ary Tree is applied from SP1 to SP3, the Joined Q-ary Tree only needs to adjust its branch at SP4, SP5, and SP6.

	SP	288 tags		576 tags		864 tags	
		2	4	2	4	2	4
<b>H</b>	<b>1</b>	0	-	0	-	0	-
<b>GMN</b>	<b>2</b>	0	-	0	-	0	-
<b>OC(I)</b>	<b>3</b>	37	-	37	-	37	-
<b>OC(U)</b>	<b>4</b>	-	55	-	53	-	53
<b>SN(I)</b>	<b>5</b>	61	-	61	-	61	-
<b>SN(U)</b>	<b>6</b>	-	89	-	89	-	89

Table 10: Actual Separating Point for Data set one, two, and three. At a specific SP, Joined Q-ary Tree will adjust its branch to either 2-ary or 4-ary tree.

## 5.2 Results

Based on the experiment results shown in table 11 and figure 7, the Joined Q-ary Tree always performed the best out of the three approaches considered, while 4-ary Tree has the worst performance, regardless of number of tags within an interrogation zone.

Table 11 demonstrates that 2-ary tree has a better performance than 4-ary tree by about 1 percent, while Joined Q-ary Tree's performance is approximately 12 percent better than the 2-ary tree. The 4-ary Tree has the worst performance out of the three approaches considered. Figure 8 illustrates the improvement in percentage of Joined Q-ary Tree compared to the 2-ary Tree and the 4-ary Tree. The percentage of improvement increases more slowly once the number of tags within the interrogation zone gets higher.

**Performances Comparison between Naive Approaches and Joined Q-ary Approach**

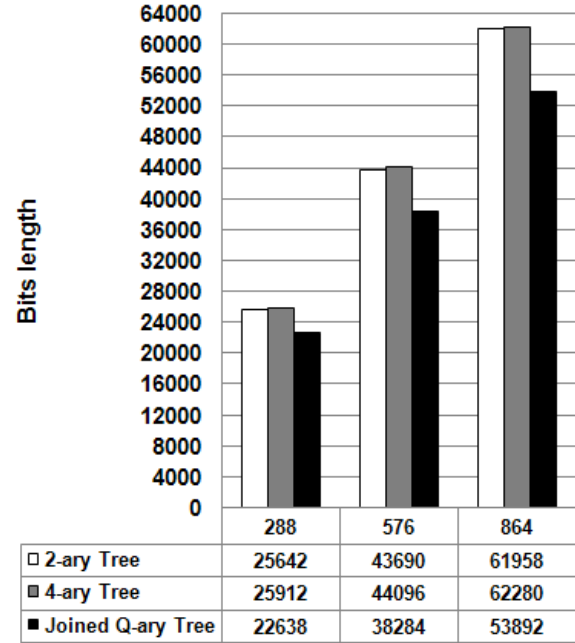


Figure 7: Performances comparison between Naive approaches and Joined Q-ary approach.

**Percentage of Improvement compare to Naive Approaches**

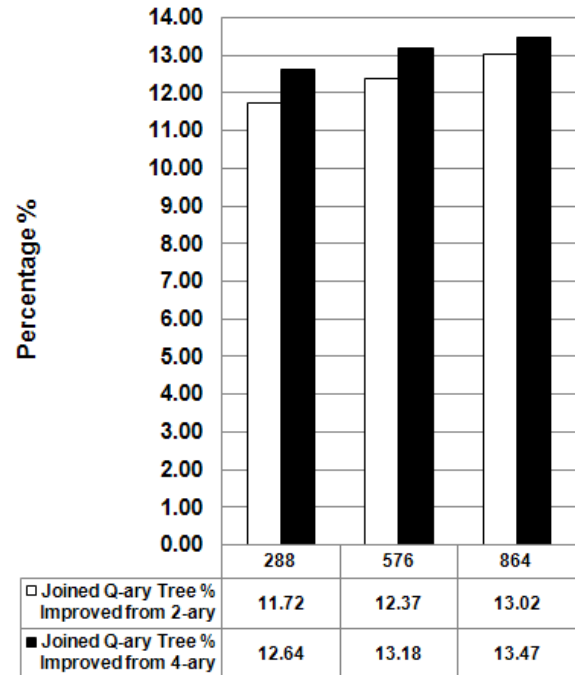


Figure 8: Percentage of improvement of Joined Q-ary Tree compared to Naive 2-ary Tree and Naive 4-ary Tree.

	2-ary		4-ary		Joined	
	% improved from 2-ary	% improved from 4-ary	% improved from 2-ary	% improved from 4-ary	% improved from 2-ary	% improved from 4-ary
288	0	1.04	-1.05	0	11.72	12.64
576	0	0.92	-0.93	0	12.37	13.18
864	0	0.52	-0.52	0	13.02	13.47

Table 11: Results of 2-ary Tree, 4-ary Tree, and Joined Q-ary Tree on three different data sets.

### 5.3 Analysis and Discussion

To further analyse and compare performances of Naive Q-ary approaches and Joined Q-ary approach, table 12 shows *Accumulative Bits Length* of each approach until all tags were identified. We can see that the number of *Bits Length* accumulates faster when the identification reached SP4. This is when EPC data became more Unique and larger *Bits Length* issued by reader were required. All three approaches have the same pattern of increment, where number of tags in an interrogation zone have no impact on the performances when EPC data were Identical. This pattern can be seen in figure 9, where *Accumulative Bits Length* of the Joined Q-ary Tree approach is displayed. There are no or little differences between SP1 to SP4 on all data sets, however, number of *Bits Length* started to increase rapidly once it reaches SP5 and SP6.

	2-ary Tree		
	288 tags	576 tags	864 tags
SP1	72	72	72
SP2	1332	1332	1332
SP3	2970	2756	2756
SP4	3682	3730	3926
SP5	17178	27642	38310
SP6	25642	43690	61958
	4-ary Tree		
	288 tags	576 tags	864 tags
SP1	80	80	80
SP2	1368	1368	1368
SP3	3024	2808	2808
SP4	3736	3776	3816
SP5	17400	27968	38536
SP6	25912	44096	62280
	Joined Q-ary Tree		
	288 tags	576 tags	864 tags
SP1	72	72	72
SP2	1332	1332	1332
SP3	2970	2756	2756
SP4	3358	3308	3348
SP5	17246	27948	38628
SP6	22638	38284	53892

Table 12: *Accumulative Bits Length* of three approaches: Naive 2-ary Tree, Naive 4-ary Tree, and Joined Q-ary Tree.

Out of the three approaches, the Joined Q-ary Tree has the slowest incremental rate between SP4 and SP6 for all data sets. The incremental *Bits Length* also slows down once number of tags within the interrogation zone gets bigger. Thus, the performance of Joined Q-ary Tree will increase for larger set of tags compared to the Naive approaches.

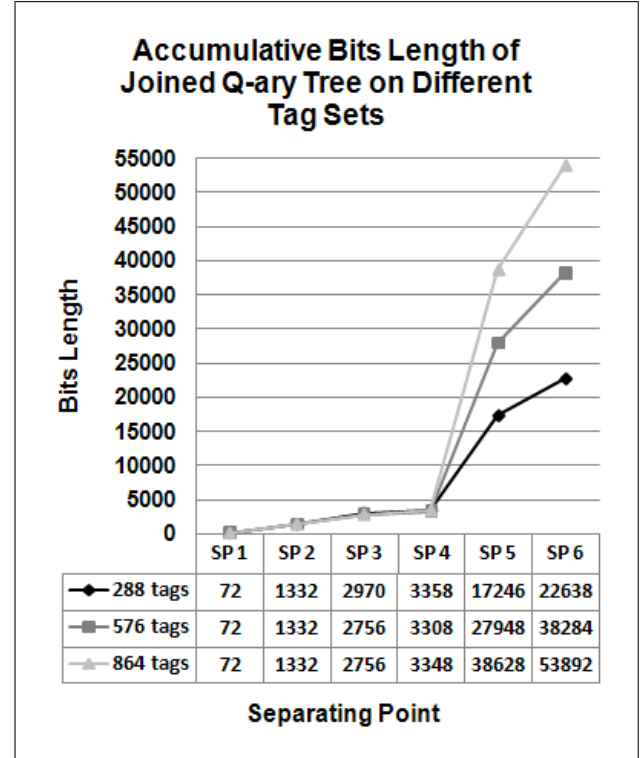


Figure 9: *Accumulative Bits Length* of three approaches: Naive 2-ary Tree, Naive 4-ary Tree, and Joined Q-ary Tree.

Since the Joined Q-ary Tree accumulated least *Bits Length* out of the three approaches, it can now be concluded that the best performing approach is Joined Q-ary Tree. Also, the number of tags within the interrogation zone have no impact on *Identical bits* of EPC data. Additionally, by applying the right Q-ary tree on specific bits of EPC, performance of Joined Q-ary Tree is improved.

## 6 Conclusion

In this study, we identified the significance of RFID tags anti-collisions and developed efficient method to minimise the *Bits Length* issued by a reader; and at the same time to ensure that 100 percent of RFID tags are identified. We proposed a “Joined Q-ary Tree”, which adaptively adjust its tree branches according to number of tags, in order to minimise *Bits Length* queried. In the experimental evaluation, we showed that the proposed method has better performance when compared to existing methods. Specifically, it requires about 12 percent less queries issued per complete identification than the existing approaches and at the same time guarantees identification of all tags.

In terms of future work, we intend to test a performance of “Joined Q-ary Tree” on different *Encoding Scheme* other than GID-96. Different pallet sizes simulation will also be inspected to determine the impact of packaging and density on quality of captures data. In addition, power consumption of readers and impact of capture effect will be observed.



## Acknowledgements

This research is partly supported by ARC (Australian Research Council) grant no DP0557303.

## References

- Abramson, N. (1970), The ALOHA System - Another Alternative for Computer Communications, in 'Proceedings of Fall Joint Computer Conference, AFIPS Conference', Houston, Texas, pp. 281–285.
- Bai, Y., Wang, F. & Liu, P. (2006), Efficiently Filtering RFID Data Streams, in 'CleanDB Workshop', pp. 50–57.
- Brusey, J., Floerkemeier, C., Harrison, M. & Fletcher, M. (2003), Reasoning About Uncertainty in Location Identification with RFID, in 'Workshop on Reasoning with Uncertainty in Robotics at IJCAI', Acapulco, Mexico.
- Carbunar, B., Ramanathan, M. K., Koyuturk, M., Hoffmann, C. & Grama, A. (2005), Redundant Reader Elimination in RFID Systems, in '2nd Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON'05)', pp. 176–184.
- Cho, H., Lee, W. & Baek, Y. (2007), LDFSA: A Learning-Based Dynamic Framed Slotted ALOHA for Collision Arbitration in Active RFID Systems, in 'Advances in Grid and Pervasive Computing Second International Conference', Vol. 4459, Springer Berlin/Heidelberg, Paris, France, pp. 655–665.
- Choi, J. H., Lee, H. J., Lee, D., Lee, H. S., Youn, Y. & Kim, J. (2008), 'Query Tree Based Tag Identification Method in RFID Systems'. [www.freshpatents.com/Query-tree-based-tag-identification-method-in-rfid-systems-dt20080508ptan20080106383.php](http://www.freshpatents.com/Query-tree-based-tag-identification-method-in-rfid-systems-dt20080508ptan20080106383.php).
- EPCGlobal (2006), 'EPCGlobal Tag Data Standards Version 1.3: Ratified Specification'. <http://www.epcglobalinc.org/standards/tds/>.
- Fishkin, K. P., Jiang, B., Philipose, M. & Roy, S. (2004), I Sense a Disturbance in the Force: Unobtrusive Detection of Interactions with RFID-tagged Objects, in 'UbiComp 2004: Ubiquitous Computing', Seattle, Washington, USA, pp. 268–282.
- Gonzalez, H., Han, J. & Li, X. (2006), Mining Compressed Commodity Workflows from Massive RFID Data Sets, in 'CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management', ACM Press, New York, NY, USA, pp. 162–171.
- Gonzalez, H., Han, J., Li, X. & Klabjan, D. (2006), Warehousing and Analyzing Massive RFID Data Sets, in 'ICDE '06: Proceedings of the 22nd International Conference on Data Engineering', IEEE Computer Society, Washington, DC, USA, p. 83.
- Jain, S. & Das, S. R. (2006), Collision avoidance in a dense RFID network, in 'WiNTECH '06: Proceedings of the 1st international workshop on Wireless network testbeds, experimental evaluation & characterization', ACM, New York, NY, USA, pp. 49–56.
- Jeffery, S. R., Alonso, G., Franklin, M. J., Hong, W. & Widom, J. (2005), A Pipelined Framework for Online Cleaning of Sensor Data Streams, Technical Report UCB/CSD-05-1413, EECS Department, University of California, Berkeley.
- Jeffery, S. R., Alonso, G., Franklin, M. J., Hong, W. & Widom, J. (2006), Declarative Support for Sensor Data Cleaning, in 'Pervasive Computing', Springer Berlin/Heidelberg, Dublin, Ireland, pp. 83–100.
- Jeffery, S. R., Garofalakis, M. & Franklin, M. J. (2006), Adaptive cleaning for RFID data streams, in 'VLDB'2006: Proceedings of the 32nd international conference on Very large data bases', VLDB Endowment, Seoul, Korea, pp. 163–174.
- Lee, C. W., Cho, H. & Kim, S. W. (2008), 'An Adaptive RFID Anti-Collision Algorithm Based on Dynamic Framed ALOHA', *IEICE Transactions* **91-B**(2), 641–645.
- Lee, S. R., Joo, S. D. & Lee, C. W. (2005), An enhanced dynamic framed slotted aloha algorithm for rfid tag identification, in 'MOBIQUITOUS '05: Proceedings of the The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services', IEEE Computer Society, Washington, DC, USA, pp. 166–174.
- Myung, J. & Lee, W. (2006a), 'Adaptive binary splitting: a RFID tag collision arbitration protocol for tag identification', *Mob. Netw. Appl.* **11**(5), 711–722.
- Myung, J. & Lee, W. (2006b), Adaptive splitting protocols for RFID tag collision arbitration, in 'MobiHoc '06: Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing', ACM, New York, NY, USA, pp. 202–213.
- Pupunwiwat, P. & Stantic, B. (2009), Unified Query Tree for RFID Tag Anti-Collision Resolution, in A. Bouguettaya & X. Lin, eds, 'Twentieth Australasian Database Conference (ADC 2009)', Vol. 92 of *CRPIT*, ACS, Wellington, New Zealand, pp. 49–58.
- Quan, C. H., Hong, W. K. & Kim, H. C. (2006), Performance Analysis of Tag Anti-collision Algorithms for RFID Systems, in 'Emerging Directions in Embedded and Ubiquitous Computing', Vol. 4097, Springer Berlin/Heidelberg, Seoul, Korea, pp. 382–391.
- Ryu, J., Lee, H., Seok, Y., Kwon, T. & Choi, Y. (2007), A Hybrid Query Tree Protocol for Tag Collision Arbitration in RFID systems, in 'MobiHoc '06: Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing', IEEE Computer Society, Glasgow, UK, pp. 5981–5986.
- Shin, J. D., Yeo, S. S., Kim, T. H. & Kim, S. K. (2007), Hybrid Tag Anti-collision Algorithms in RFID Systems, in 'Computational Science ICCS 2007', Vol. 4490, Springer Berlin/Heidelberg, Beijing, China, pp. 693–700.
- Vogt, H. (2002), Efficient Object Identification with Passive RFID Tags, in 'Pervasive '02: Proceedings of the First International Conference on Pervasive Computing', Springer-Verlag, London, UK, pp. 98–113.
- Wang, Z., Liu, D., Zhou, X., Tan, X., Wang, J. & Min, H. (2007), 'Anti-collision Scheme Analysis of RFID System', *Auto-ID Labs White Paper*. <http://www.autoidlabs.org/single-view/dir/article/6/281/page.html>.

# Structures in Collaborative Tagging

## An Empirical Analysis

A. Fani Marvasti<sup>1</sup>D.B. Skillicorn<sup>2</sup>

<sup>1</sup> Mathematics and Computer Science Department  
Baha'i Institute for Higher Education, Iran Email: [amin.fani@bihe.org](mailto:amin.fani@bihe.org)

<sup>2</sup> School of Computing  
Queen's University, Canada  
Email: [skill@cs.queensu.ca](mailto:skill@cs.queensu.ca)

### Abstract

It is widely believed that users choose meaningful tags; that the combinations of these tags from multiple users helps to elicit meaning for tagged objects; and that implicit communities can be discovered among the users who use a particular tag or tag a particular object. We examine data on tagging practices from the Delicious web site ([delicious.com](http://delicious.com)) and find that there is little support for any of these beliefs. Although users individually do seem to have a small set of tags that they use in a controlled and effective way, they also use very large sets of other tags in a much more haphazard way. These poorly managed tags obscure much of the collective sense making and implicit community structure. We derive some suggestions for improving collaborative tagging systems.

**Keywords:** social computing, tagging, information retrieval, collaborative sensemaking

### 1 Introduction

Collaborative tagging systems allow users to label any web object, typically a page but possibly also an image, audio, or video file, with a set of strings, called *tags*. Usually, these labels are then visible to others and the objects can be retrieved by the original tagger, and usually by others, using tags as search terms.

The use of tags allows objects to be categorized in any way that is useful to the individual doing the tagging. In contrast, hierarchical organization of information requires decisions about the best single place in the hierarchy where that information should be categorized. Tags are therefore sometimes considered as a substitute for taxonomies.

There are a number of widely-held assumptions about collaborative tagging, some of them embedded in the business plans of the businesses that provide collaborative-tagging sites. These assumptions are:

- Users label documents with carefully-chosen and meaningful tags that reflect some semantic aspect of the documents, or of the relationships between the individual users and the documents. Each user is creating a mapping from the document, and its content and properties, to an appropriate set of strings, in such a way that the user can, at some future time, generate the inverse function with high probability. In other

words, the tag is chosen so that, when the user wants to retrieve the document again, s/he will be able to infer or remember one of the tags required to find the desired document.

This implies that the set of tags should be parsimonious and that individual tags should be well-separated in meaning. If they are to add functionality above that of search engines, they should also reflect aspects of the documents or their properties that are orthogonal to the content, and perhaps also hard to infer automatically. In other words, the greatest potential benefit is when a tag captures a human decision about the 'meaning' of an document. (Rashmi (Sinha 2005), for example, has proposed a cognitive model that describes the advantages of tagging over categorization, and proposes how humans might choose tags.) Users receive recompense for making such decisions by exploiting the tags of other users to improve their own retrieval success.

- Users act collectively to make sense of the content and meaning of each document. Each user makes a small intellectual contribution by choosing 'good' tags and, in return, is able to use the well-chosen tags of others to find information that might be difficult to discover by conventional search. In some systems, tags already used to label a particular document are displayed so that a user can reinforce the existing opinions about tags that are good; or suggest new ones to the community of those interested in this document. Smith (Smith 2008) explains tagging as a way of bridging personal and community knowledge; Chi (Chi 2008) claims that users choose tags both for themselves and for others; and Weick *et al.* (Weick et al. 2005) describe tagging as collective sense making. For this collective process to work, each user must maintain and use a mental model of how to label a new document effectively, as described in the previous point, and these models must generally agree.
- The set of users who tag a particular document, and the set of users who use a particular tag have other similarities, so that these common connections allow the detection of communities among users that might be hard to discover in other ways. The implicit graph connecting users, tags, and documents can be used to carry out clustering, query expansion, and enhanced search experience, for example by generating recommendations for one user based on the knowledge supplied implicitly by others. John and Seligman (John & Seligmann 2006) and Rashmi (Sinha 2005) build social networks from the way in which documents have been tagged.

Here we report empirical studies of tagging behavior at one of the popular collaborative-tagging sites, Delicious ([delicious.com](http://delicious.com)). In general, our results provide little support for these assumptions about how collaborative tagging is done. In particular, we observe that:

- Users tag in a haphazard way. The tags used often have little connection to the document being tagged, individuals use very large numbers of similar tags, and users make many mistakes in tagging. This suggests that they do not use a parsimonious mapping of content to tags.
- Users do not appear even to reuse their own tags successfully. They tend to generate tags similar to those they have already used, apparently having forgotten the precise form of their previous tags. Clearly retrieval cannot be effective when this happens.
- Given that individual users fail to reuse their own tags well, it is not surprising that there is little evidence that one user's tags help other users to search more effectively.
- Although use of the same tag or tagging of the same object obviously indicates some kind of common interest, the implicit communities this suggests seem to be very diffuse. This is partly because of the use of multiple similar tags which diffuses what might be a single community into a multitude of weaker communities. Discovering any communities that do exist seems difficult.

Our tentative conclusion is that collaborative-tagging systems are ineffective for most users, even for their individual information-retrieval needs. As a form of collaborative sense making, it is even less clear that collaborative tagging achieves very much. There do seem to be some users who are tagging documents with many tags, presumably altruistically to make them accessible to as many other users as possible. However, there is little evidence that this pays off in better access by other users.

It is not clear whether this failure of collaborative tagging is related to novelty, so that tagging quality will improve as users gain experience; or whether it is related to interface design, so that better information would produce better tagging; or whether tags are so individual and taste-based that they do not generalize well.

There is a need for interview-based analysis of tagging behavior to validate these tentative conclusions. Meanwhile, it seems unlikely that deeper understanding of social processes can be derived from today's collaborative-tagging data.

## 2 Related Work

Several empirical studies of tagging systems and tagging behavior have been carried out. Golder and Huberman (Golder & Huberman 2006) examined the different kinds of tags in use and classified them as having the following functionalities: identifying what a document is about; identifying what kind of document it is; identifying who owns it; refining existing categories; identifying qualities or characteristics; self-referencing by the tagger; and organizing tasks. Smith (Smith 2008) also suggests that some tags are intended to attract search engines. Golder and Huberman also analyzed the temporal properties of tags, showing that some tags remain popular while others enjoy bursts of popularity. They also show that the distribution of tags associated with a particular document tends to become stable, so that the relative

frequency of each tag becomes constant (and novel tags are then only rarely applied to it).

Wetzker *et al.* (Wetzker et al. 2008) analyzed Delicious between 2004 and 2008 and observed that the site is heavily biased towards technology-related content. They also showed there is a power-law relationship between users and documents tagged. The top 1% of users generate 22% of all tagging events, and the top 10% generate 62%. There is also a power-law relationship between tags and documents tagged – 700 tags account for 50% of all tagging events. They also indicated that they suspected that the most active Delicious users were actually spamming, that is tagging with the intent of driving traffic to certain web pages or sites, but we find no evidence to support this.

Chi and Mytkowicz (Chi & Mytkowicz 2008) also used data from Delicious, and examined tags and documents using entropy-based measures. Their most significant finding was that the mutual information between tags and documents is linearly decreasing with time, so tags are becoming less and less useful as search terms. They also find that the rate of increase in the total number of tags in use has flattened, something that we do not observe for individual users.

Halpin *et al.* (Halpin et al. 2007) did a similar analysis over a longer time period, and found that power laws are characteristic of activity in this domain. They also showed that tag distributions become stable, supporting the results of Golder and Huberman, and showed how these distributions evolve in the period before stability.

Zeng and Li (Zeng & Li 2008) experimented with using the structure of the tripartite graph of users, tags, and documents to draw conclusions about user interests that could be used to make recommendations. They conclude that tags can be useful to improve user similarity calculations, but doing so requires very sophisticated clustering techniques.

Begelman *et al.* (Begelman et al. 2006) used co-occurrence similarity to cluster tags and were able to show that some plausible clusters could be obtained. One example, the cluster based around “health” includes: “shopping”, “research”, “nutrition”, “food”, “diet”, “fitness”, “workout”, “running”, “article”, “science”, “esport”, “sport”, “product”, “life”, “lifehack”, “howto”, “gtd”, “reference” and “tip”. While some of these tags are clearly health-related, many of the others have only a tenuous connection to it, suggesting that tag similarity is quite diffuse.

Santos-Neto *et al.* (Santos-Neto et al. 2006) analyzed tagging behavior in CiteULike and Bibsonomy, tagging systems that specialize in bibliographic records. They found a significant correlation, for each user, between number of tags and number of tagged objects, perhaps because finding a bibliographic reference requires a unique key. They also observed a large number of singleton users with no overlap of tags or tagged objects with anyone else.

## 3 Properties of Tags

We carried out a number of empirical studies using data collected from the [delicious.com](http://delicious.com) website during the second half of 2008. Delicious has more than 5 million registered users who have tagged more than 150 million objects.

Delicious allows registered users to tag any web object they encounter. When this happens, sets of popular and recommended tags are displayed from which the user can choose, but arbitrary new tags may also be created. Each tag is constrained to be a single (blank delimited) string. Various communication mechanisms within the site are possible. For

Set	Mean	Std Dev	Max	Min	Mode
Tags $T$	46.97	11.39	75	20	44
Keywords $K$	24.92	9.52	37	2	30

Table 1: Statistical properties of tags and keywords extracted from the chosen documents.

Mean	Std Dev	Max	Min	Mode
7.2%	3.7%	16.2%	1.4%	3.8%

Table 2: Statistics of Jaccard similarity between tags and extracted keywords

example, when an object is tagged, the event can be sent to a user's friends within an explicitly defined personal network.

The simplest functionality of such a site is that it replaces individual browser bookmarking, with the added advantage that tagged objects are accessible from anywhere on the Internet, not only the browser from which they were first encountered. It seems clear that many users use Delicious only for this purpose.

Tags, and the objects they label are public on the Delicious web site, unless explicitly hidden, so anyone, not just a registered user, can search using a tag (to get the corresponding objects) or a url (to see the corresponding tags). Lists of the most popular and recent tags over the whole site are also available.

Delicious is just one of a number of popular sites that permit tagging. For example, Youtube ([www.youtube.com](http://www.youtube.com)) specializes in video, Flickr ([www.flickr.com](http://www.flickr.com)) in images, Last.fm ([last.fm](http://last.fm)) in music, and CiteULike ([www.citeulike.org](http://www.citeulike.org)) in bibliographic references.

### 3.1 The relationship of tags to content

A dataset consisting of 100 documents and their tags was extracted from the Delicious.com web site. Each document had been tagged by at least 50 users, so that the set of tags applied is likely to have converged to a relatively stable set (as predicted by (Golder & Huberman 2006)). The documents were also required to be primarily textual so that, for example, pages of links to video or audio were excluded. The process was seeded by beginning from users who had used tags on the *popular* list.

Each document was processed to remove ephemeral content, for example inserted advertisements, and the html markup was removed to leave plain text. The plain-text documents were then processed using Yahoo's Term Extraction web service, which extracts keywords and phrases. (It is also advertised as an *automatic* way to construct tags from documents.) The Yahoo service extracts phrases as well as single words, while Delicious tags are only allowed to be single strings, which causes some discrepancies, although minor.

Statistical properties of the sets of tags and keywords extracted from this set of documents are given in Table 1. It is clear, and slightly surprising, that the number of tags is substantially larger and more variable than the extracted keywords. Note especially that the *minimum* number of tags for any of these documents is 20.

We compute the similarity between the set of tags and the set of keywords for each document using Jaccard similarity:

$$\text{sim}(T, K) = \frac{|T \cap K|}{|T \cup K|}$$

The results are given in Table 2. It is clear that the overlap between tags and keywords is extremely

Percentage	Type
50%	<i>Purpose or type words</i> such as "reference", "howto", "tips", "tools", "tutorial", and "free-ware".
19%	<i>Modified keywords</i> . This includes both plurals and other modifications of actual keywords, or strings built by concatenating words (because of Delicious's limitations), for example, "word-pressplugins".
11%	<i>Acronyms and non-English words</i> . This includes many common abbreviations, for example "ux" and "aussie"; as well as words from other languages that were not recognized as keywords by the term extractor.
7%	<i>URLs and format terms</i> including file types such as "mp3" or "pdf".
4%	<i>Times and dates</i> , for example when a product was released or a book published.
3%	<i>Process words</i> , for example "toread", "todo".
3%	<i>Adjectives</i> , for example "interesting", or "cool".
2%	<i>Punctuation and function words</i> , for example ",", "and", and words such as "and" and "or". Some of these are clearly caused by misunderstanding Delicious's rules for forming tags.
1%	<i>Misspellings</i> of plausible keywords.

Table 3: Percentage distribution of non-keyword tags

low, that is tags are not simply descriptions of the tagged document's content.

One possible explanation is that users do not tag documents with content-based tags because searching on content is straightforward with mainstream search engines. However, at present there is no way to search on *both* content and meta-information encoded in other kinds of tags, so it would seem reasonable to use content tags extensively.

### 3.2 The role of tags

The typically 90% of tags that did not correspond to keywords were examined manually and categorized as shown in Table 3.

The most striking thing, of course, is how rare it is to label documents with tags that could be considered abstractions of their content, such as hypernyms. Golder and Huberman (Golder & Huberman 2006) suggest that this is a common form of labelling, but we find little evidence for this.

Labelling with purpose tags is extremely useful since inferring purpose algorithmically is difficult. This category is by far the most useful for sense making. Providing tags that describe document content in other languages and using common abbreviations is also useful. However, many abbreviations and variations are somewhat idiosyncratic. It is conceivable that algorithmic techniques such as stemming might help to condense the tag sets in use, but our impression is that the variety is so great that this would be extremely challenging.

It is also clear that users would prefer a richer language for tagging that would allow multiple-word

tags. However, addressing this deficiency would not necessarily be an improvement because the number of possible multiple-word phrases to describe a particular concept is large, making it even harder to get agreement among users.

Process and adjectival tags show a lack of awareness of the global context of tagging, because such tags rely on context to be useful. Labelling a document as “toread” is useful to an individual only if it is also tagged with a unique identifier, and generating such a unique identifier in an unknown global environment is difficult. There is little sign that such identifiers are being used.

Similarly, an adjective is an individual opinion which is unlikely to have useful weight in an environment with 5 million users.

The results suggest that people often use tags as a way of adding personal notes or non-obvious details about the content for their own consumption – they use tags for browsing and organizing, rather than information retrieval, similar to the way they put their personal photos in an album and add marginal notes.

## 4 How Individuals Tag

We now consider tagging behavior in more detail by looking at the set of tags used by an individual.

### 4.1 How individuals use tags

We extracted data for 100 random users, each of whom had used at least 100 tags. The frequencies of use for each of the tags for each user were then plotted, sorted in descending order of frequency of use. Results for four users are shown in Figure 1, truncated after the 100 most-frequent tags. The distributions are very similar for all of the users investigated so we did not investigate a larger set of users.

The discontinuities in frequencies, indicated by the circles, separate tags with qualitatively different usage patterns. There is typically a small set of tags, typically less than 10, that are heavily used. There are then one or two sets that are used at lower frequencies. The broad shape of these curves is consistent with the power-law behavior observed by others; but the discontinuities suggest that a refinement is needed. The boundaries between these regions suggest that there may be two different mental mechanisms at work for handling tags – some concept of a working set of well-remembered tags that users maintain and which perhaps captures their primary interest(s); and a much larger set of tags that are recollected and applied in a more diffuse way.

### 4.2 How individuals tag documents

We extracted data for 500 random users and considered how many documents they each tagged. The results are shown in Figure 2. There are a number of users with both very large numbers of tags used and very large numbers of documents tagged. The bottom left-hand corner of the figure is expanded in Figure 3, showing that users use many tags and label many documents. In general, for a given user there are more tagged objects than there are tags, but not by much. This suggests that tags are reused to label multiple objects, but not much on average. Given the distribution of tag frequencies observed in Figure 1, this suggests that some tags are used heavily, but most are used only a very few times.

It is clear from this figure that many users are extremely active taggers. The number of tags used by an individual ranges from 6 to 23,887, with mean 2009; and the number of documents tagged by an

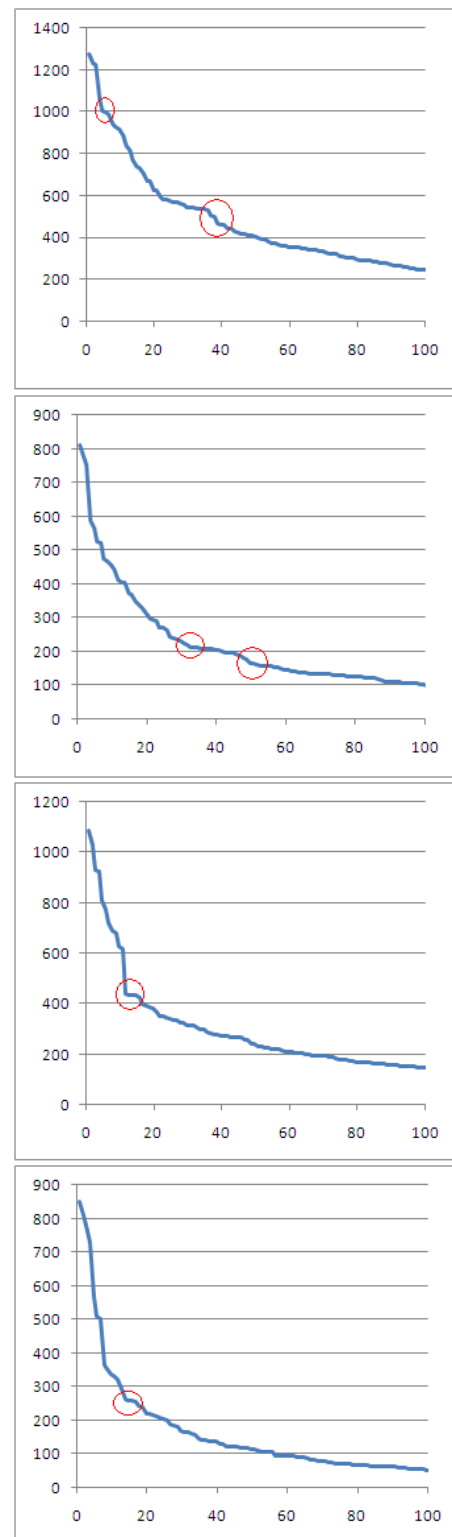


Figure 1: Tag frequencies in descending order for four selected users

individual ranges from 1 to 53,116, with mean 4593. 12% of individuals use more than 4000 tags to label more than 7000 documents; and 61% use more than 2000 tags to label more than 3000 documents. The correlation between number of tags used, and number of documents tagged is low, around 0.05.

It is possible that an account at Delicious represents more than one individual, and that this might account for the number and variety of tags. However, this seems unlikely given the existence of a distinct ‘knee’ in curves such as those in Figure 1, and the apparent existence of a small number of tags that are used consistently.

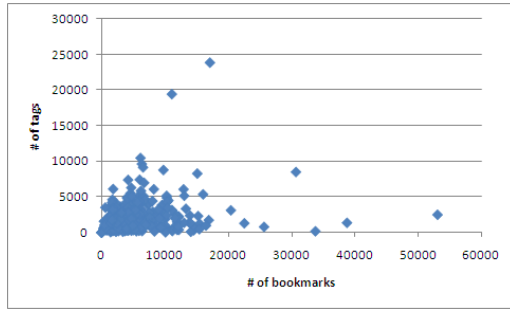


Figure 2: 500 users plotted by number of bookmarks (tagged documents) and number of tags each used

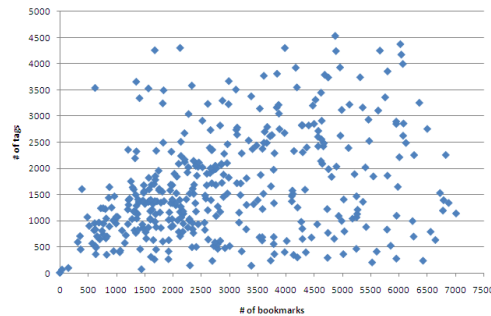


Figure 3: 500 users plotted by number of bookmarks (tagged documents) and number of tags each used (zoomed)

It has been suggested that users with either high rates of tag use, or tagging large numbers of documents are spamming, possibly algorithmically. We investigated this question by examining, manually, the tags and documents of the individuals at the high end of the spectrums of both. There are 15 individuals who have labelled more than 10,000 documents. In all cases, the tags they used appeared to relate to the content of the pages they tagged, and the sets of documents tagged were not from a limited set of domains. In other words, it seems implausible that they were tagging with any malign purpose such as directing traffic to any particular documents or sites.

There are several discernible characteristics to the tag usage of these heavy-tagging individuals:

- They use personalized tags, acronyms, and numbers heavily;
- They tend to include urls and domains as tags;
- They use many redundant tags (for example, “weblog”, “blogs”, “blog”, “blogging”, “blogger”, and “weblogs” for the same document);
- They make many typographical and spelling errors; and
- They use both English and non-English words.

It seems implausible in the extreme that individuals maintain a mental list of thousands of tags that they believe will be useful for their own retrieval. However, it is possible that these users regard themselves as acting altruistically, labelling documents in as many ways as they can imagine so that they can be retrieved by other users in many different possible ways. This provides some evidence for a social dimension to tagging. Nevertheless, the actual usefulness of this approach is limited by the number of errors made in the tags themselves, and the low specificity of the tags used. The potential benefit seems small compared to the amount of effort made.

### 4.3 Document similarity based on users

To compute the similarity of documents tagged for each user we use the same set of 500 users whose total number of tagged documents were considered earlier. For each user, we select the 50 most-recently tagged documents. For each pair of documents, similarity is computed as the Jaccard similarity using both the extracted keywords and tags used for each document. This captures both the inherent similarity (based on the content of each pair of documents) and the user views of similarity (based on the tags applied to each pair of documents). The average similarity between the documents tagged by each user are shown in Figure 4, together with the standard deviations (sorted into increasing order of average similarity).

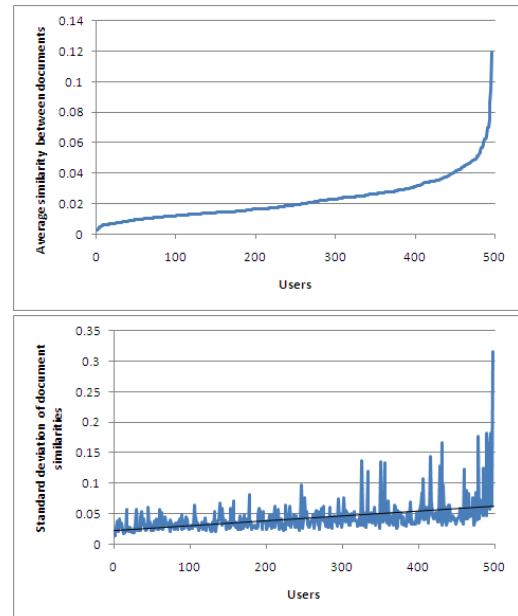


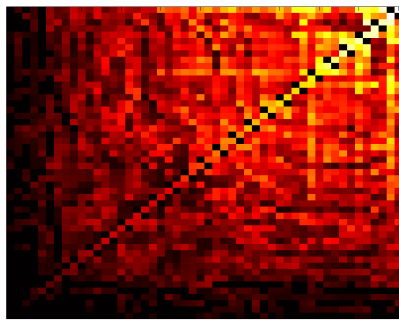
Figure 4: Average and standard deviation of document-document similarities for 500 users

This figure shows that the average overlap among documents tagged by any user is rather small; in other words, users have diverse sets of interests. For 90% of the users, the similarity among documents is below 4%, while for the remaining 10% it only reaches a maximum of 14%. This suggests that knowledge of which users have tagged a document provides little useful information for finding, for example, related documents.

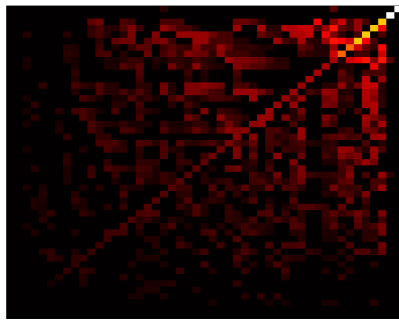
Some examples of the similarity matrices for particular users are shown in Figure 5, with lighter shades reflecting greater similarity, and with rows and columns alternately sorted to the top right-hand corner. For User 54, there is a great deal of weak similarity among tagged documents, but very little clustered structure. In contrast, User 209 has two well-defined clusters of documents, but the remaining documents are even less related than those of User 54. User 128 lies in between these extremes.

The *maximum* overlap between documents tagged by each user, seen in Figure 6, shows some interesting structure. There are three groups of users in the figure with identical similarities (flat regions of the curve in the figure): a block of 40 users with maximum document similarity of 33%, a block of 22 users with maximum document similarity of 66%, and a block of 50 users with maximum document similarity of 100%. The similarity of documents is computed using both the tags and keywords associated with each document. For most documents, the size of the tag set is roughly twice the size of the keyword set. The most likely explanation is that, for the first group,

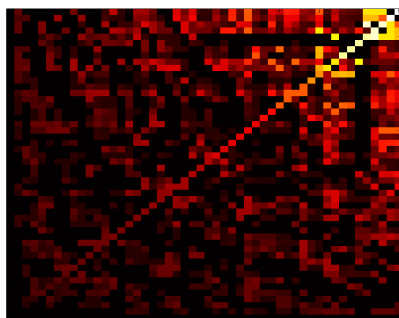




[User 54]



[User 128]



[User 209]

Figure 5: Document-document similarity matrices for a selection of users

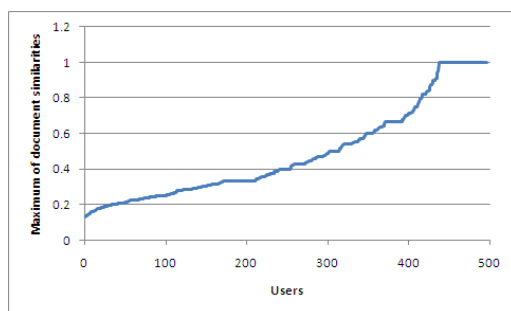


Figure 6: Maximum document-document similarity over all users

there are documents where the keywords match but the tags do not; for the second, there are documents where the tags match but the keywords do not; and for the third group, there are documents where both match.

#### 4.4 Tagging over time

To examine the way in which users tag more deeply we now consider how an individual uses tags over time.

We collected a dataset of two sets of users: ten who have tagged 100 or fewer documents ever; and ten who have tagged more than 1500 documents. The users were selected randomly from among those who used tags on the popular bookmark list. Users of the first kind are assumed to be naive or beginning users, while those of the second kind are assumed to be experienced.

We then retrieved, for each user, data about the tags used on consecutive days when any activity was recorded, and computed the Jaccard similarity between successive days. The results, for a typical inexperienced and experienced user are shown in Figure 7 (note the different scales on the axes in the two graphs).

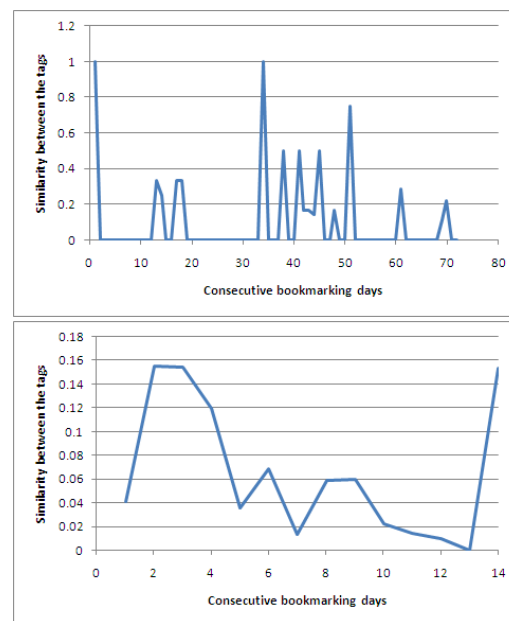


Figure 7: Jaccard similarity between tags used on consecutive days for an inexperienced and experienced user

The typical usage of an inexperienced user is that there is either a lot of overlap between two consecutive days, suggesting that an area is being explored; or very little overlap, suggesting that the user has moved on to a new topic of interest. It is not surprising that such a user can remember and reuse tags effectively, since they have only a small set of tags in use.

In contrast, experienced users have little overlap from one day to the next. This seems surprising, since we might expect that such users have a well-defined set of interests, and would continue to tag new documents related to these interests, so their tag sets would be stable and overlaps from day to day would be large. Manual inspection of such users suggests that the low overlap is the result of imperfect recollection of which of a possible set of tags they have previously used. For example, a single user used “addon”, “addons”, “add-on”, and “add-ons”; while another used “answer”, “answers”, “answers,”, and “answers)”, both within a short time period. This reinforces the earlier result suggesting that users maintain a small list of tags that they can remember and use appropriately, and a much larger set that they remember only hazily and so use with less control.



## 5 Communities of Users

We now consider the question of whether either tags or documents provide any extra structure that connects users into communities. In other words, are the users who all use a particular tag similar in other ways. The existence of such communities derived from tagging practices could be used to provide recommendations, or to expand queries by including tags known to be related (via communities) to the tag being used for search.

To compute the similarity of users using a particular tag, we collected the top 500 most-popular tags, and selected the 50 most-recent users to have used each of these tags. We now consider how similar each of these groups of 50 users is.

There are several useful measures of user-user similarity that could be computed for such a set of users: the similarity of the set of documents they have tagged; the similarity of the content of the set of documents they have tagged; or the similarity of the tag sets they have each used. The first is problematic because there is not a 1-1 relationship between documents and the urls that describe them (for example, because of the use of tiny urls); and the second is expensive and limited by the effectiveness of content extraction. So we use the third, and compute the Jaccard user-user similarity for each of the 50 users of each tag.

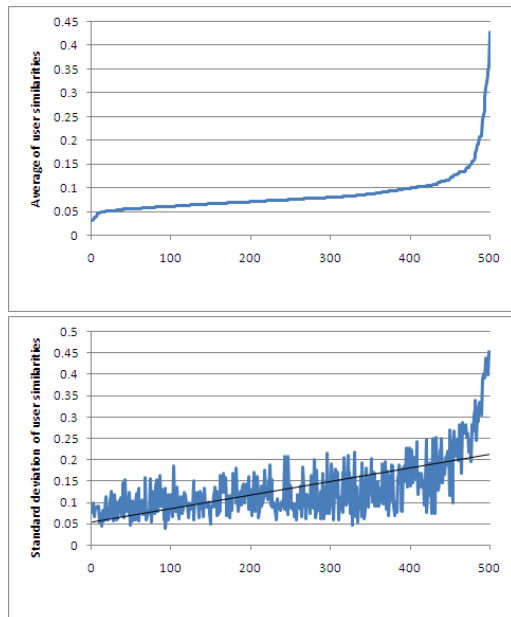


Figure 8: Average and standard deviation of user-user similarities for the first 500 most-popular tags

The average similarity for the 500 tags is shown in Figure 8, together with the standard deviations. The list of tags is sorted into ascending order of user-user similarity. For approximately 90% of the tags, the user-user similarity is low, so there is little evidence of user communities associated with these tags. For the remaining 10% of tags, the user-user similarity is as high as 42%, but the standard deviations of these similarities also increase substantially, suggesting that there is not a systematic relationship – the similarity is highly tag-dependent. The maximum similarities per tag were also computed and, for all but one tag, there are at least two users with identical tag sets.

Some of the actual user similarity matrices are shown in Figure 9. As before, these matrices have been sorted alternately by row and column to move the largest entries to the top right-hand corner.

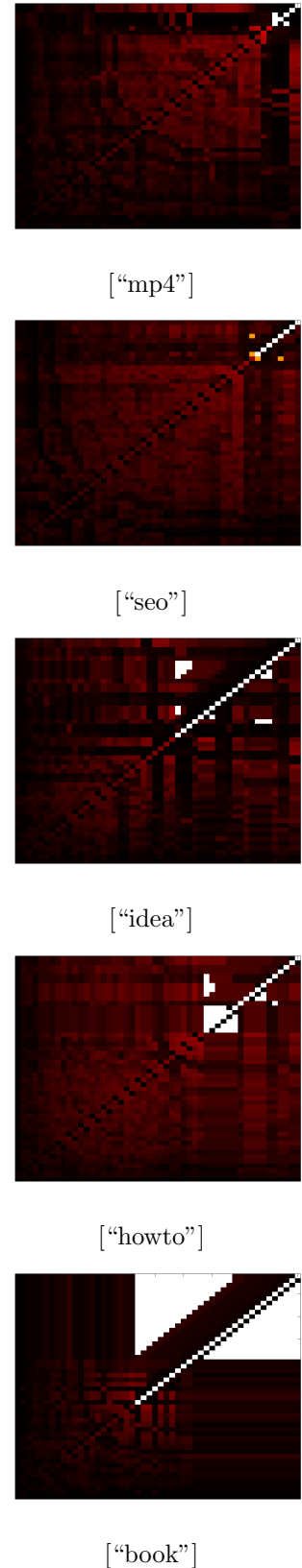


Figure 9: User-user similarity matrices for a selection of tags; lighter shades mean greater similarity, coloured bars indicate (weak) clusters

There are only a few small clusters among the users who have used each tag; the only exception we discovered is the tag “book” which has one large cluster. Note especially that purpose tags, which might be expected to be useful to others, do not have substantially more structure than content tags. These figures suggest that there is useful knowledge about implicit user communities present in tagging data,

but that it will be hard to extract. This is partly because the useful data is overwhelmed by less-useful frequent data. Just as the presence of highly advertised bestsellers makes collaborative recommendation difficult, the presence of popular but non-useful tags may make implicit community discovery difficult.

## 6 Conclusions

The business plans of commercial collaborative tagging sites, and much academic work on collaborative tagging has assumed that tags are chosen carefully based on a desire and expectation to use them for later retrieval; that tags can synthesize information encoded in existing tags with each new user's view of an object to produce a collective synergy of understanding; and that common use of tags, and tagging of common documents reflects the presence of underlying implicit communities of interest.

We find very little evidence, at present at least, to support these assumptions. The primary problem seems to be the way in which tags are actually chosen. Far from being carefully chosen to reflect the content, meaning, and context of an object, tags seem to be selected on the fly, with little thought. There is some evidence that users maintain a very small list of tags (of size perhaps no larger than 10) that they are able to use in a careful, effective, and consistent way. However, they then seem to use very large sets (thousands) of tags in a haphazard way, labelling similar objects encountered even very close in time with different tags. This makes it highly implausible that users are able to retrieve previously tagged objects even using their own tags.

The collaborative aspect of tagging also seems to be only weakly supported. It is clear that many users are using tags to provide information that is not directly present in objects such as meta-information or translations of the content into other languages. What they do *not* appear to do is to use tags to describe the content at different level of abstraction (e.g., hypernyms of the content words). The potentially useful information from tagging is largely spoiled, or at least diffused, by poor and inconsistent choices of the actual tags. It seems that some users are labelling with very large numbers of tags in an effort to make objects as easy as possible to find, but it is not clear that this effort is worth the improved access.

This suggests that tagging performance might be improved by restricting both the form and number of tags that a user could apply to any object. For example, forcing a maximum number of tags (say 5) would encourage clearer thought about the *best* tags for a given object, and would also encourage conventions such as always using singular rather than plural noun tags. More forceful presentation of existing tags would also help to develop conventions about what makes a good tag. (On the other hand, Chi and Mytkowicz suggest that users be presented with existing tags and encouraged to choose novel tags.) Thus interface design in tagging systems might well improve the performance and usefulness of such systems.

We also found little evidence of user communities among those who use any given tag, and little evidence of similarities among the documents tagged by a given user. It appears that each individual tends to have wide interests, as expressed in what they tag. Trying to find communities via these commonalities will only be effective if powerful techniques are used, since the communities are so weak.

These results suggest that both technology and social constraints can be used to improve collaborative

tagging. Better interfaces that force quality of tags rather than quantity would probably help. More direct feedback about how others are using tags would also probably improve the quality of tagging. There is clearly a demand for richer tagging languages, but it is not clear that this would improve the situation because the increased expressiveness also creates more ways to express essentially the same tag.

It is conceivable that the deficiencies we have observed are because tagging is a new phenomenon, and users have not yet become accustomed to how to use it well. However, Delicious is dominated by technologically sophisticated users, so it would be surprising if this were the only issue.

## References

- Begelman, G., Keller, P. & Smadja, F. (2006), Automated tag clustering: Improving search and exploration in the tag space, in 'WWW2006 Workshop on Collaborative Web Tagging'.
- Chi, E. (2008), 'The social web: Research and opportunities', *IEEE Computer* pp. 88–91.
- Chi, E. & Mytkowicz, T. (2008), Understanding the efficiency of social tagging systems using information theory, in 'Hypertext '08', Pittsburgh, pp. 81–88.
- Golder, A. & Huberman, B. (2006), 'The structure of collaborative tagging systems', *Journal of Information Science* **32**(2), 198–208.
- Halpin, H., Robu, V. & Shepherd, H. (2007), The complex dynamics of collaborative tagging, in 'The International World Wide Web Conference'.
- John, A. & Seligmann, D. (2006), Collaborative tagging and expertise in the enterprise, in 'WWW 2006'.
- Santos-Neto, E., Ripeanu, M. & Lanitchi, A. (2006), Tracking user attention in collaborative tagging communities, in 'Proceedings of International ACM/IEEE Workshop on Contextualized Attention Metadata'.
- Sinha, R. (2005), 'A cognitive analysis of tagging', [rashmisisinha.com/2005/09/27/a-cognitive-analysis-of-tagging/](http://rashmisisinha.com/2005/09/27/a-cognitive-analysis-of-tagging/). Accessed December 2008.
- Smith, G. (2008), *Tagging : People-powered Metadata for the Social Web*, New Riders Press.
- Weick, K., Sutcliffe, K. & Obstfeld, D. (2005), 'Organizing and the process of sense making', *Organization Science* **16**(4), 409–421.
- Wetzker, R., Zimmermann, C. & Bauckhage, C. (2008), Analyzing social bookmarking systems: A del.icio.us cookbook, in 'Proceedings of the ECAI 2008 Mining Social Data Workshop', IOS Press, pp. 26–30.
- Zeng, D. & Li, H. (2008), How useful are tags? - an empirical analysis of collaborative tagging for web page recommendation, in 'Social Computing Workshop SOCO2008', pp. 320–330.

# Extended Boolean Retrieval for Systematic Biomedical Reviews

Stefan Pohl

Justin Zobel

Alistair Moffat

NICTA Victoria Research Laboratory,  
Department of Computer Science and Software Engineering  
The University of Melbourne,  
Victoria 3010, Australia  
{spohl,jz,alistair}@csse.unimelb.edu.au

## Abstract

Searching for relevant documents is a laborious task involved in preparing systematic reviews of biomedical literature. Currently, complex Boolean queries are iteratively developed, and then each document of the final query result is assessed for relevance. However, the result set sizes of these queries are hard to control, and in practice it is difficult to balance the competing desires to keep result sets to a manageable volume, and yet not exclude relevant documents from consideration.

Ranking overcomes these problems by allowing the user to choose the number of documents to be inspected. However, previous work did not show significant improvements over the Boolean approach when ranked keyword queries based on terms in the Boolean queries, review title, research question or inclusion criteria were used.

The extended Boolean retrieval model also provides ranked output, but existing complex Boolean queries can be directly used as formal description of the complex information needs occurring in this domain. In this paper we show that extended Boolean retrieval is able to find a larger quantity of relevant documents than previous approaches when comparable (or greater) numbers of documents are inspected for relevance.

**Keywords:** Information retrieval, extended Boolean retrieval,  $p$ -norm, effectiveness, systematic review, biomedical.

## 1 Introduction

Web search is one of the most prominent Information Retrieval (IR) applications. Typical question-answering scenarios are well supported by ranking highly the documents that not only look relevant by their content, but also receive external support such as by incoming links and anchor text references. In these applications, looking at one or a few of the highest ranked documents might be sufficient, and if it is, the search process can be stopped. Commercial web search engines are optimized for this scenario and much IR research is focused on improving performance in the top, say 10, results.

However, if the objective is to carry out a comprehensive review for a particular topic, search cannot be stopped after finding a few relevant documents. In particular, reviews aim for very broad coverage of a topic, and seek to minimize any bias that might arise as a result of missed or excluded relevant literature. But the typical tensions in IR continue to apply, and if more relevant documents are to be found, more irrelevant documents will also need to be

inspected. In the biomedical domain, systematic reviews of the whole corpus of published research literature (the largest collection, MEDLINE, currently indexes more than 17 million publications) are used to provide medical practitioners with advice to assist their case by case decision-making. To seed the reviews, complex Boolean queries are used on different citation databases to generate a set of documents which are then triaged by multiple assessors. In this domain, it becomes crucial to find as much of the relevant literature as possible for any given level of effort, because each item of overlooked evidence adds to the possibility of suboptimal outcomes in terms of patients' health-care.

The traditional Boolean retrieval model has been studied intensively in IR research. While it has straightforward semantics, it also has a number of disadvantages, most notably the strictly binary categorization of documents, and the consequent inability to control the result set size except by adding or removing query terms. For example, it is often the case that too many, or too few, or even no documents are returned, and no matter how the query terms are juggled, the "Goldilocks" point might be impossible to attain. In contrast, the broad adoption of ranking principles based on bag-of-word queries, and the resultant ability to order the set of documents according to a heuristic similarity score, means that for general IR applications users can consciously choose how many documents they are willing or able to inspect. Now the drawback is that bag-of-word keyword queries do not offer the same expressive power as Boolean queries do. Although extensions to the Boolean retrieval system have been suggested that produce a ranked output based on Boolean query specifications, they have not been broadly adopted for practical use – perhaps because, to date, simple keyword queries have typically been able to produce similar results, and, for lay users, are easier to generate.

Although ranking has the advantage of identifying a monotonically increasing total number of relevant documents as more documents are inspected, typical IR ranking functions face the difficulty that their ranking is dependent on properties of the whole collection, and can thus be difficult to reproduce, or even understand. Reproducibility helps in assessing review quality, and is thus often stipulated as a key requirement of comprehensive reviews [Sampson et al., 2008]. But if ranked queries are used, reproducibility can only be assured if all aspects of the computation are reported, including term weights and within-document term frequencies. With Boolean queries, all that is required is publication of the query that was used, together with the date or other identifying version numbers of the collections it was applied to. Moreover, previous work did not show improved retrieval results with ranked keyword queries compared to complex Boolean queries [Karimi et al., 2009].

In this paper, we show that, for the searching undertaken for the purposes of systematic reviews, an extended Boolean retrieval model finds more relevant documents



Figure 1: A typical query tree, showing different types of operators. Many of the details have been omitted.

than other current alternatives when typical numbers of documents are inspected, where typical is in terms of the result set size achieved by the initial complex Boolean queries specified by the researchers undertaking the review. Ranked results based on Boolean queries facilitate iterative Boolean query refinement through assessment of the top retrieved results and allows pure Boolean result sets to be extended if there is capacity to inspect more documents. As was noted already, finding more relevant documents with similar effort reduces the risk of missing important evidence that could affect decisions made by a medical practitioner.

## 2 Background

Evidence-based medicine aims to apply the latest published, scientific evidence in medical practice [Davidoff et al., 1995]. Systematic reviews of the current and prior literature are an essential tool to thoroughly and objectively survey the literature in order to address a specific research question. To minimize bias, the researcher's goal is to identify almost all of the publications reasonably related to the topic under review. Significant amounts of time are dedicated to the initial search phase, usually in the order of weeks or months [Zhang et al., 2006]. Search is performed using complex Boolean queries through interfaces such as PubMed<sup>1</sup> or Ovid<sup>2</sup>, in multiple bibliographic databases like MEDLINE or EMBASE, each containing millions of citation entries. These entries usually contain titles and abstracts, and meta-data such as publication year, author(s), language, and manually indexed annotations in form of Medical Subject Headings (MeSH)<sup>3</sup>. After a final search strategy has been determined, all returned documents are triaged for their relevance based on their title, abstract and finally full-text. Other methods are also used to extend the set of relevant documents, such as following citations forward and backward, and

hand-searching of conference proceedings. The evidence in each document in regard to the research question the review addresses is appraised, extracted, and finally synthesized [Higgins and Green, 2008]. To make the search reproducible, the search strategies used to locate the set of studies that are formally cited in the review are published along with the collated view of that literature that is provided by the review.

Queries are complex in multiple dimensions, notably operator richness, structure and size.

First, the operators used are beyond those in traditional pure Boolean systems. As well as binary conjunction (AND) and disjunction (OR) operators and their  $n$ -ary versions, field restrictions might be employed, or proximity operators, or wildcard term expansions, or MeSH terms and their so-called "explosion". All of these extensions must be supported within the syntax and semantics of the query language. For example, a query like

wom?n AND exp \*Genomics/

matches all documents containing both a wildcard expansion of wom?n, such as woman and women, and with a main focus (the MeSH \* qualifier) on Genomics or any MeSH heading below Genomics in the MeSH hierarchy, such as Proteomics (the MeSH exp operator).

Second, Boolean logic allows operators to be nested deeply, to express arbitrary concepts and the relations between them. Most queries follow a basic structure close to conjunctive normal form (CNF) that is generally referred to as faceted search in the literature [Hersh, 2008]. Semantically close terms, phrases, or MeSH headings are connected in disjunctions (using OR) which are then combined in a top-level conjunction (using AND). Often, further conjuncts are added that are based on meta-data, and act as filters corresponding to the inclusion or exclusion criteria of the review.

Finally, queries can become very long, and it is usual to split queries into different sub-parts and then subsequently combine the partial results by references to previous query line numbers. In total, queries typically consist of between a dozen and a hundred or more query lines, each containing terms, basic concepts, and simple operators; and

<sup>1</sup><http://www.ncbi.nlm.nih.gov/pubmed/>

<sup>2</sup><http://www.ovid.com/>

<sup>3</sup><http://www.nlm.nih.gov/mesh/>

when written down as a single fully-expanded Boolean expression, can involve quite significant complexity.

Figure 1 shows parts of the query tree for one typical query. The range of operators that are used can be seen, and also the large number of nodes that are present as part of the query.

Because the queries are intended to be (re-)used later as filters against which newly published documents can be screened so that reviews can be periodically updated, they should generalize well and not over-fit to match on a possibly large fraction of documents retrospectively known to be relevant. In particular, while there is always a single Boolean query that returns exactly the set of all documents already known to be relevant, namely a disjunction of unique phrases drawn from each relevant document, this level (or any approximation of it) of over-fitting must be guarded against. Instead, the published queries are just the final statements used to generate the set of reviewed documents out of which a large fraction of the cited ones were drawn, and are thus formal descriptions of an information need, expressed in terms of Boolean logic.

## 2.1 Motivation

Boolean queries are best employed in data retrieval scenarios in which it is known beforehand what records exist, and which ones of them are to be retrieved. For example, searching through an email archive for a message with a particular document attached to it might proceed on the basis of knowing an approximate date of the email, the topic of the document, and the name of the sender of the email. With such cues available, Boolean fielded search can usually be relied on to locate the required item; and the searcher is satisfied immediately that a single email/document combination has been located.

Databases are a typical example of such controlled environments, and the manual indexing in bibliographic databases – and hierarchical classification schemes such as Dewey and Library of Congress – seek to add this feature to unstructured textual databases. But not all information needs can be foreseen, and providing a suitable set of indexing terms that covers all possible eventualities is tantamount to adding the whole of the document as index terms. This is why Boolean retrieval is also applied to unstructured textual data. However, the ambiguity of free-text, and the lack of a controlled vocabulary leads to the well-known precision-recall trade-off in IR, which notes that only a fraction of the documents in the returned set are actually relevant (the *precision* of that set of documents); and only a fraction of all relevant documents are in the returned set (the *recall* of that set of documents). These two competing requirements – high precision, so that the searcher’s time spent examining documents is used to best effect; and high recall, so that the majority of the relevant documents are identified as part of the search – can be balanced by adjusting the query. For example, adding conjuncts to a query is likely to increase precision but decrease recall; and adding disjuncts is likely to increase recall but decrease precision. In the limit, if all documents are returned (by a disjunctive query containing terms of all documents), then recall is 1.0, but this is an unsatisfactory situation. Similarly, a query in which no documents are returned technically has precision of 1.0, but is equally useless (unless there are no relevant documents in the collection).

The Boolean retrieval model has a long history, but a number of main problems are repeatedly reported in the literature. For instance, novices and lay users may find Boolean queries difficult to formulate [Frants et al., 1999]. While this is an issue in general IR, a search intermediary, such as a librarian that knows the database, is usually part of a review team in this domain [McGowan and Sampson, 2005]. A bigger problem is that documents are only

Query Tree	# Docs	Fraction
AND	2,935	0.56
OR	258,560	0.67
headache	37,758	0.22
“muscle cramp”	1,617	0.11
...		
OR	62,337	0.78
...		
human	10,885,697	1.00
...		

Table 1: Case study of a query with low overall success. The final column shows the fraction of the known relevant documents (based on all techniques, not just this one query) that are identified by that subexpression.

differentiated into two stark groups: those that match the query and are retrieved and inspected, and those that do not match and are not retrieved and thus never viewed, regardless of whether or not they are relevant. Moreover, in a typical conjunction-of-disjunctions query, documents are not retrieved if only one conjunct evaluates to false, and treated as if every conjunct had evaluated to false; and are retrieved regardless of whether all of the terms in each disjunct match, or just one. The documents in the retrieved set are then all treated identically and returned in either random order (database record number, for example), or according to some secondary sort criterion such as (reverse) date of publication. Since all documents in the final Boolean result set are inspected, it may seem that the presentation order is unimportant. But worth noting is that the creation of complex queries requires iterative refinement and assessment of query quality, and that this preliminary work is of necessity done on a subset of the documents returned by each trial query. In contrast to sampling the whole result set, probing the top results of a ranking based on relevance to the query could possibly reduce these costs. Finally, as has already been noted, the size of the result set of a Boolean query is largely out of the user’s control.

It is thus not surprising that analysis of the query performance in this domain shows that many of them actually do not find all documents finally included in reviews [Dickersin et al., 1994, Martinez et al., 2008]. Error analysis for low-success queries revealed situations such as depicted in Table 1. Although a reasonable fraction of the documents known to be relevant match with each of the conjuncts comprising the overall query, overall success is (naturally) not greater than that of the least successful conjunct. This is the price paid to reduce the result set size for the individual conjuncts to a reasonable size, and is typical of the patterns observed for Boolean queries. In particular, while not all of the relevant documents contain all the required query concepts, the query would in part have been constructed so as to generate a result set of manageable size, perhaps 1,000 to 2,000 documents.

While a document containing a term of each of the concepts only once is included in a Boolean result set and is possibly only marginally relevant, a document containing frequent appearances of terms of multiple concepts, but completely missing one concept as expressed in the query, is strictly excluded. Additionally, issues such as typographic or indexing errors, or use of abbreviations or unanticipated synonyms are more likely to be influential in citation databases than in full-text collections. Ranking solves these problems and allows users to choose consciously how much effort they are willing to invest into the search.

Although extended Boolean retrieval has been shown to improve retrieval results compared to strict Boolean evaluation, the complexity in specifying a Boolean query does not pay off if similar retrieval results can be achieved

with ranking and simple keyword queries. However, previous work in the medical domain has shown that ranked retrieval using the information need descriptions at hand did not lead to higher performance [Karimi et al., 2009]. One possible explanation is that the complex information needs in this domain cannot be expressed as bag-of-words queries amenable to currently used ranking functions.

## 2.2 Related Work

For years, the use of Boolean queries has been deeply embedded in process guidelines for biomedical systematic review search [Higgins and Green, 2008]. As a consequence, there is much literature investigating issues around them. Dickersin et al. [1994] found that, although the sought documents are present in MEDLINE, the sensitivity of queries is unsatisfactory even if only meta-data is searched. Beahler et al. [2000] conclude that Boolean search is not enough because binary matching is insufficient due to indexing errors. Also, search in multiple databases has been suggested to alleviate this problem [Avenell et al., 2001].

Few papers suggest ranking as a solution to the low answer relevance density of Boolean queries. Martinez et al. [2008] use different textual information from the systematic review as a query to get an initial ranking which is then refined using a pseudo-relevance feedback technique. Karimi et al. [2009] found that loosening the strictness of the queries combined with ranking of the result set is able to achieve higher relevance fractions and outperform each individual method. Still, the low relevance densities attained suggest that for medical abstract searching the highest realistic aim can only be to find as much relevant literature as possible for a given effort, rather than finding every relevant document. However, using textual descriptions as keyword queries with typical ranking functions has recently been shown to perform poorly [Bendersky and Croft, 2008]. It is thus surprising that descriptive ranked queries perform as well as Boolean queries, although key concepts are formally present in the Boolean query specifications. Although the previous work considered ranking, none of the studies used the Boolean queries themselves for this purpose.

Cohen et al. [2006] estimate the usefulness of an approach based on classification of the citations in the Boolean result set to screen out documents that are likely to be irrelevant. To train a classifier, they assumed half of the documents to be judged. Naturally, this limits the possible improvement. While this approach might be able to reduce costs associated with judging documents, it cannot find additional relevant document not in the Boolean result set. However, if applied to filtering newly published documents for their relevance to systematic reviews, more relevant documents could be found than with Boolean filters. Shojania et al. [2007] give evidence that a large fraction of systematic reviews need to be regularly updated.

Extended Boolean models generate ranked output from Boolean query specifications. In the past, many extended Boolean models have been suggested that vary in the ranking function used, the arity of the operators, and in support for query weights. Simple fuzzy-set models [Radecki, 1979] seek to extend the pure Boolean model to support non-binary term weights, but effectively use the same score function as the pure Boolean model. More sophisticated functions are used in the Waller-Kraft [Waller and Kraft, 1979], Paice [Paice, 1984],  $p$ -norm [Salton et al., 1983] and Infinite-One models [Smith, 1990]. Lee [1995] gives a good overview of these models.

The sparseness and age of the literature suggests that extended Boolean models have been not as successful as ranking functions based on keyword queries. This seems

plausible if similar or better results can be achieved with simple keyword queries which are easier to create. However, it is not clear that this holds in the biomedical domain where complex information needs are to be satisfied that partly include strict inclusion criteria based on meta-data.

## 2.3 Extended Boolean Retrieval Models

Retrieval models can be characterized based on the assumed query and document representations, the retrieval function, and the form of output. While Boolean models are set-oriented and retrieve only documents that satisfy a Boolean constraint, vector-space models consider documents and queries as vectors. In the latter, both documents and queries are represented as *bags-of-words*, and real-valued similarities are calculated between them and used to rank the documents, a useful presentational device. The former allows representation of more complex information needs in the query. We choose to elaborate on the extended Boolean retrieval model of Salton et al. [1983], also known as  $p$ -norm model, selected as a basis for exploration because the ranking formula of this model has been shown to have desirable properties that promote good rankings [Lee, 1994].

To deal with the nested structure of Boolean queries, ranking functions for Boolean queries are defined for each operator separately, working with a tree structure that reflects the structure of the original Boolean expression. The final similarity score is calculated recursively, working from the leaves back to the root of the query tree, applying the corresponding score aggregation formula at each internal node.

The basic operators in each Boolean query are conjunctions (AND) and disjunctions (OR). Salton et al. [1983] define the ranking score  $s$  for  $n$ -ary disjunctions as

$$s_{\text{OR}}(w_1, \dots, w_n) = \left( \frac{1}{n} \sum_{i=1}^n w_i^p \right)^{1/p},$$

and for  $n$ -ary conjunctions as

$$s_{\text{AND}}(w_1, \dots, w_n) = 1 - \left[ \frac{1}{n} \sum_{i=1}^n (1 - w_i)^p \right]^{1/p},$$

where the  $w_i$  values are the weights of terms in the interval  $[0, 1]$  when the children are leaves of the tree, or are the scores of the sub-trees also in the range  $[0, 1]$ , as appropriate to the tree structure below this node; and where  $p$  is a parameter in the interval  $[1, \infty)$ .

The simplest initial choice for document term weights is to assign binary weights at the leaves of the tree, with 0 indicating the absence and 1 the presence of a term. More complex weightings are possible for document as well as the query terms. The choice made for parameter  $p$  then determines a particular ranking function within a continuum. In particular, when  $p=1$ , a simple inner product document-query similarity function is used, and when  $p=\infty$ , depending on the choice of term weights, either fuzzy-set or strict Boolean retrieval is performed. If the operand scores are regarded as weights of a vector and defined not to be negative, then the formulas become  $p$ -norms of vectors based on the operand scores, normalized to the interval  $[0, 1]$ . (Hence the name of the technique.)

In pure Boolean systems, it is sufficient to implement conjunctions and disjunctions as binary operators because associativity,  $\text{op}(a, \text{op}(b, c)) \equiv \text{op}(\text{op}(a, b), c)$ , allows  $n$ -ary versions of the operator to be handled via any application of multiple binary operators,  $\text{op}(a, b, c) \equiv \text{op}(a, \text{op}(b, c))$ . However, in the extended Boolean system associativity does not hold, and terms at higher levels in the operator



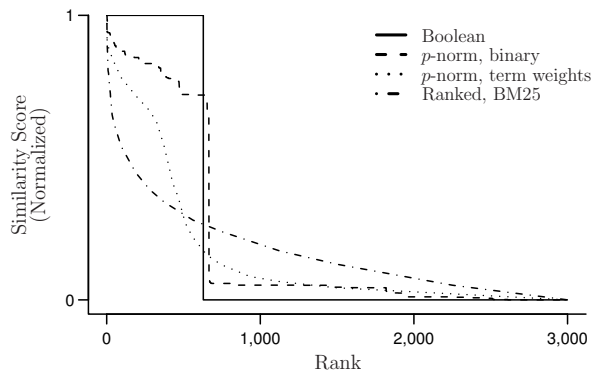


Figure 2: Ranked similarity score distribution for one example query through to rank 3,000, with scores normalized to  $[0, 1]$ . Note that the domain is actually discrete, and that the score levels at different ranks have been connected for presentational purposes only.

tree have a larger influence on the final query-document similarity score than do lower ones. To circumvent this problem, systems should implement  $n$ -ary versions of the operators [Lee, 1994], and queries should be written to make appropriate use of them.

Depending on the number of possible initial term weights, and on the structure of a query, only a limited number of output levels are possible. That is, the degree in which ranking is performed is restricted, and an ordered list of unordered sets is returned, rather than a continuous ranking. This becomes apparent looking at similarity scores of queries for different models, as illustrated in Figure 2. To break similarity score ties a second order criterion can be used, typically the same criterion used for the Boolean result set, for example, reverse chronological. In this sense, continuous term weights might be preferable, presuming some principled method for assigning them can be determined. On the other hand, a study of the score distribution in conjunction with the query structure can be valuable in determining document quantities to which specific operands, or a particular number of conjuncts and disjuncts on the same query tree level match. Recall that the structure of the queries at hand and the  $p$ -values used influence the final similarity score, with operators on higher levels in the query tree dominating the score computation.

An interesting observation follows from the example depicted in Figure 2. Based on query-document similarity scores, documents might be clustered into two groups, those with high and low similarity scores. Overall, extended Boolean retrieval using binary term weights tends to follow loosely the shape of the Boolean retrieval function. Thus, the large step in the consecutive similarity scores derives from documents that either satisfy all or only some conjuncts in the conjunction at the top of the query tree. In these two groups, the number of matching disjunctions then determines variation in the scores. However, note that the score gap does not happen at exactly the same rank as for Boolean evaluation, meaning that some documents indeed are able to compensate for less influential, unsatisfied conjuncts. It would be wasteful to exclude these few documents from consideration, because they can be very informative in order to refine the query based on relevancy of these documents.

### 3 Experiments

We chose to use Lucene<sup>4</sup> as indexing engine due to the available MEDLINE parser as part of the LingPipe project<sup>5</sup>. We then implemented our own query parser,

<sup>4</sup><http://lucene.apache.org/java/>

<sup>5</sup><http://alias-i.com/lingpipe/>

query operators and query evaluators that directly access the inverted lists, assign scores to documents, and return either a set or a ranking of the highest scoring documents. For expansion we used a recent version of MeSH as of 2008. Boolean result sets are returned in reverse chronological order because this is the default ordering criterion imposed by most Boolean search systems. That is, within a set of equal-score items, the recent ones are returned preferentially over older ones, on the assumption that age degrades usefulness.

We compare our results against two Boolean baselines, our own implementation, and Ovid, the interface to which these medical queries are usually submitted. We entered the queries into this online system, and retrieved the PubMed identifiers of each returned document. Furthermore, analogous to the work presented by Martinez et al. [2008], we used zettair (version 0.9.3) with the Okapi ranking function [Robertson et al., 1995] and default settings, to generate keyword query baselines based on review title, research questions, inclusion criteria and the plain terms used in Boolean queries. These queries are denoted as the TRC formulation in our experiments.

### 3.1 Data

Relevance judgments and queries for 15 systematic reviews have been made available by Cohen et al. [2006]. Documents are considered to be relevant if they were included in the final review, without regard to how they were found (that is, irrespective of whether they were found by querying the collection, by following citations, or by hands-on involvement via conference proceedings or personal knowledge). This reduces bias in favor of the Boolean queries used. The relevance judgments in this dataset have been restricted to MEDLINE documents published between 1993 and 2003 (inclusive) because they were the ones originally made available via the TREC Genomics Track [Hersh et al., 2004].

As a collection, we took all active (not withdrawn) abstracts of MEDLINE as of late 2008, and a subset consisting of 4,515,866 abstracts published in the year range corresponding to the relevance judgments. We then removed from the relevance judgments any references to documents not in the collection that we had formed. This reduced the number of relevant documents by around 10%, on average; a small penalty in order to create a test environment in which (if it could but exist) a “perfect” query was one which located all of the known relevant documents. Additionally, we used the queries and relevance judgments for 13 additional systematic reviews from a testset made available by Karimi et al. [2009] as a hold-out set, to validate our findings.

### 3.2 Reproducing results

A key requirement in this domain is reproducibility, which is why the Boolean queries are published together with the reviews that they contribute to. However, it has been noted that the published queries contain typographic errors [Sampson and McGowan, 2006]. Additionally, meta-data such as MeSH headings are used which are under continuous refinement. Fortunately, many authors also document the result set sizes achieved for each component line in their query script, which could be used as a guide during query debugging. Even so, differences in document parsing, in the actual document sets used, in query parsing and transformation (for instance term mapping or error correction), means that there can be variation between systems executing the same Boolean query. This might be one of the reasons why it is recommended that multiple databases are to be searched, even though there is considerable overlap between databases.



Query	# Nodes	Result set size		
		Ovid	Reprod.	Overlap
1	66	1,597	1,457	1,452
2	35	979	884	884
3	120	353	342	342
4	71	735	629	629
5	47	3,856	3,805	3,727
6	37	1,377	1,340	1,337
7	77	306	248	248
8	38	286	271	271
9	80	742	823	629
10	26	282	258	258
11	57	1,108	1,106	1,106
12	72	718	682	682
13	44	2,460	2,331	2,104
14	51	413	372	372
15	409	823	801	801
Mean	112	1,069	1,023	989

Table 2: Number of query tree nodes, and Boolean result set sizes using Ovid and our own system (column “Reprod.”) for each of fifteen test queries. The final column shows the cardinality of the intersection between the two results sets.

We fixed all obvious errors in the queries (such as unmatched terms), and changed MeSH headings that had been subsequently refined so that they referred to the appropriate subheadings, and reached the point where we reasonably reproduce results that we got by using Ovid directly with our own system (Table 2). Note that the differing result set sizes are typical of the issues with Boolean queries. The searches using Ovid were performed in mid 2009 and any documents not in our collection have been removed from those result sets.

### 3.3 Evaluation measure

The typical evaluation measures used in IR are precision and recall, measured either across the whole of a Boolean result set, or at some particular depth in the ranking, if the system assigns scores to documents. In order to incorporate more information about the distribution of relevant documents over ranks, more complex evaluation measures can also be used for rankings, for example, MAP, NDCG or RBP (see Moffat and Zobel [2008] for a summary of these). However, all of these measures place considerable emphasis on the top of the ranking; and in the medical domain all documents in the final result set are likely to be inspected for relevancy. Hence, a set-based measure is to be preferred.

In the particular application under consideration, very patient searches are undertaken, and costly evaluation processes are tolerated in the interests of research quality. It is thus not unusual for thousands of documents to be inspected once a query has been finalized, and whereas typical web-search quality can be regarded as being measured by (say) precision over the first five documents in the ranking, here we wish to evaluate a system according to whether, over a substantial answer set, a comprehensive set of answers has been located. In addition, that evaluation measure should not pay terribly much regard to exactly where in the ranking (or answer set) the relevant documents occur.

Precision is easy to measure, since it is based on the documents encountered when traversing through an answer set or ranked list. On the other hand, recall (at least, in its formal definition, see Zobel et al. [2009] for discussion of recall-like metrics) is retrospective, and can only be computed after all of the relevant documents in a collection have been identified. If exhaustive inspection of documents is impossible (and in all realistic scenarios that

is the situation), any estimate as to the number of relevant documents has some element of uncertainty associated with it. It is certainly true that if many different techniques are applied independently and the results pooled, then a greater number of relevant documents are likely to be found, and hence the number of relevant documents not found must have decreased. But even so, the best that can be said is that the total number of relevant documents that have been identified provides both a lower bound on the actual number of relevant documents, and an upper bound on the number of relevant documents that any particular system can, in an experiment, be expected to identify.

In the experiments reported below, we measure the number of relevant documents found at different ranks based on the size of the Boolean result set and for different absolute ranks, respectively; and then standardize them across queries by dividing by the number of known relevant documents for each query before computing an overall average. However, we refrain from denoting this quantity as being “recall”, since the entire document collection has not been inspected, and actual recall scores will thus likely be lower than the values reported. Note also that most valuable for systematic review search is to find more relevant documents in the (say) top 500 to 2,000 ranks. The Boolean queries are likely to have been targeted to return up to that many documents, because at these depths in the ranking the number of documents to inspect is of manageable size.

### 3.4 Implementation

We replaced higher order operators, such as explosions of MeSH headings by disjunctions of all headings below that entry in the MeSH hierarchy; terms containing wildcards by disjunctions of all terms in the dictionary that match the given pattern; and MeSH entries themselves by term lookups in a particular field containing the MeSH headings of the documents.

All complex operators were mapped to the three basic Boolean operators in order to execute them in an extended Boolean fashion. Because we wanted to profit from the extended Boolean interpretation of all restrictive operators, we chose to do this also with phrases and proximity operators. They became simple conjunctions and thus allowed documents to still be ranked highly if one term of a phrase did not occur in the document. For efficiency reasons, MeSH explosions and wildcard term expansions have always been implemented as Boolean disjunctions. This did not affect retrieval effectiveness because often the expansions are spurious terms due to spelling or parsing errors that occur only in a few documents but increase query size excessively. Otherwise, they are flection variations, synonymous terms or hyponyms and should be treated equally if the hypernym is queried.

The original queries were not intended to be executed in an extended Boolean sense and hence, no care has been taken to specify Boolean connectives in their binary or  $n$ -ary form – the associativity property of Boolean logic means that any equivalent specification leads to the same result. But this does not hold for the ranking functions used in extended Boolean models. Here, the structure of the query tree determines the influence of operands and thus the final similarity score. To normalize the queries making them more amenable to extended Boolean evaluation, we logically flattened cascades of (often binary) conjunctions or disjunctions, respectively, into their  $n$ -ary equivalents. For instance, a query

(a or (b and c)) and (d and (e or (f or g)))

would be transformed to

and(or(a, and(b, c)), d, or(e, f, g))

Id System	Number of relevant documents (normalized)								
	at absolute ranks					at ranks relative to $B_q$			
	100	300	1,000	3,000	10,000	$0.25B_q$	$0.5B_q$	$B_q$	$2B_q$
<i>a</i> : $p$ -norm <sub>BIN</sub> , $p=9$	0.18 <sup>cd</sup>	0.40 <sup>c'd'e</sup>	0.62 <sup>c'd'e</sup>	0.72 <sup>c'd'</sup>	0.79 <sup>c'd'</sup>	0.28 <sup>cd'</sup>	0.44 <sup>de</sup>	0.61 <sup>be'</sup>	0.69 <sup>b'c'd'e</sup>
<i>b</i> : $p$ -norm <sub>TF-IDF</sub> , $p=9$	0.16 <sup>c</sup>	0.34	0.57	0.70 <sup>c</sup>	0.77 <sup>c'd'</sup>	0.22	0.43	0.59 <sup>e</sup>	0.63
<i>c</i> : Boolean Reprod.	0.09	0.29	0.48	0.58	0.59	0.16	0.34	0.59 <sup>e</sup>	
<i>d</i> : Boolean Ovid	0.10	0.27	0.49	0.60	0.61	0.14	0.31	0.58 <sup>e</sup>	0.61
<i>e</i> : TRC queries	0.15	0.27	0.48	0.66	0.81 <sup>c'd'</sup>	0.20	0.29	0.41	0.58

Table 3: Number of relevant documents found for different systems, averaged over 15 AHRQ queries and normalized by the number of known relevant documents. We measured at absolute ranks and ranks based on multiples of  $B_q$ , the size of the reproduced Boolean result set for query  $q$ . We also tested for statistical significant differences between all systems using a one-sided Wilcoxon signed-rank test and report improvements at the 0.05 (Id) and 0.01 (Id') levels, respectively.

Id System	Number of relevant documents (normalized)								
	at absolute ranks					at ranks relative to $B_q$			
	100	300	1,000	3,000	10,000	$0.25B_q$	$0.5B_q$	$B_q$	$2B_q$
<i>a</i> : $p$ -norm <sub>BIN</sub> , $p=9$	0.10	0.22	0.36 <sup>c'</sup>	0.45 <sup>c'd</sup>	0.54 <sup>b'c'd'</sup>	0.19	0.26	0.34	0.40 <sup>c'</sup>
<i>b</i> : $p$ -norm <sub>TF-IDF</sub> , $p=9$	0.08	0.17	0.31	0.41	0.47 <sup>c'd</sup>	0.12	0.24	0.34	0.37
<i>c</i> : Boolean Reprod.	0.06	0.15	0.27	0.36	0.36	0.13	0.22	0.36 <sup>de</sup>	
<i>d</i> : Boolean Ovid	0.05	0.16	0.29 <sup>ac</sup>	0.37	0.37	0.13	0.21	0.30	0.37
<i>e</i> : TRC queries	0.11	0.19	0.30	0.40	0.48 <sup>c</sup>	0.16	0.21	0.27	0.33

Table 4: Performance on a hold-out set of 13 AHRQ queries.

### 3.5 Experimental setup

As a ranked retrieval baseline, we reproduced the results reported by Martinez et al. [2008], and confirmed their findings. More evidence as part of the query lead consistently to better results. Thus, we only report the results for TRC queries that are based on the concatenation of title, research question and inclusion criteria. These performed significantly better than queries using only part of the information, or the terms in the Boolean queries.

For the  $p$ -norm model, we tried a range of  $p$ -values suggested in the literature. The best performance was recorded with  $p=9$ , but retrieval effectiveness was not very sensitive to this parameter. At the extremes, use of  $p=\infty$  returned the same results (using binary term weights) as obtained with a strict Boolean implementation, and  $p=1$  performed worse than using a modern keyword ranking function naively on the terms in the Boolean query. We consider binary term/leaf weights as input to the  $p$ -norm computation; and also term/leaf weights based on the TF-IDF formulation proposed by Salton et al. [1983].

### 3.6 Results

Table 3 summarizes the results, where each reported value is the fraction of the known relevant documents determined by the given ranking depth, averaged over the query set. The superscript letters give the result of significance tests against other methods listed in the table. For example, the second entry in the table's first row, 0.40<sup>c'd'e</sup>, indicates that on average 40% of the known relevant documents are within the top 300 answers for the  $p$ -norm method, and that this is significantly better at the 0.01 level than the averages in the same column in rows *c* and *d* of the table, and significantly better at the 0.05 level than the average value listed in row *e* of the table.

First, note that our Boolean baseline system (denoted "Reprod.") gives performance similar to the Ovid results, further confirming the accuracy of our baseline.

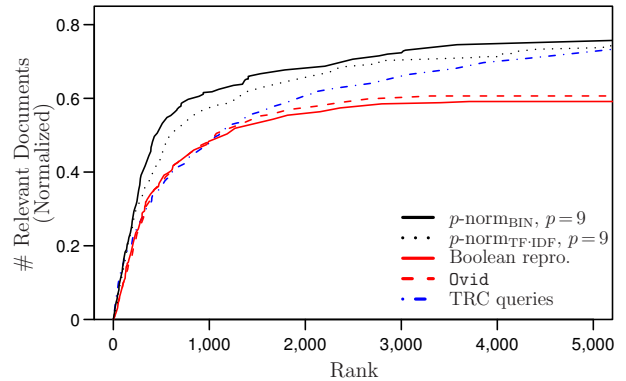


Figure 3: Average number of relevant documents (normalized by number of known relevant documents) for 15 AHRQ queries as a function of ranking depth.

Second, if absolute ranks are considered (the left-hand half of the table), the extended Boolean system with binary term/leaf weights always finds significantly more relevant documents than the other methods. This is due to averaging over multiple Boolean queries that have varying output sizes. While queries with large Boolean result sets have an increased density of relevant document at the beginning of their ranking, queries with small result sets might not be able to produce that many documents. This exemplifies the advantage of ranking, but does not compare the performance of the system for each individual query.

The third observation is drawn from the right-hand half of Table 3, which reports results at answer set sizes calculated as multiples of the size of the result sets for the original Boolean queries, rather than at fixed depths. When a quarter of the Boolean set size is inspected, on average double that many relevant documents can be found using the  $p$ -norm model with binary term weights.

This suggests the benefit of ordering Boolean result sets by relevance rather than recency. Although we could not show any improvement when the same number  $B_q$  of documents for each query are inspected, we find significantly more documents when double that many documents are viewed. Because result set sizes have not consciously been chosen and might thus be smaller than the capacities of the review team allow for, it seems to be a valuable investment to be able to look behind the cut-off that pure Boolean execution enforces.

Fourth, while the broader, ranked TRC queries are able to catch up at retrieval sets of size 5,000, inspecting that many documents is likely to be infeasible for small review teams, and other retrieval techniques would be preferred (compare Figure 3). Weighting terms individually in the extended Boolean system could not compete with binary weights though. This may be a result of inappropriate weight assignments, and is a direction we plan to explore further – it seems counter-intuitive that starting with binary term/leaf values can be superior to starting from real-valued ones, provided those values are well chosen.

Overall, we conclude that the extended Boolean retrieval results are significantly better than the Ovid ones in almost all situations.

Because we developed our system with the test queries and tuned the  $p$ -value to that set of queries, we also generated a new testset containing 13 further queries, and applied the same experiment. Table 4 gives the outcomes. The queries in this dataset identify smaller fractions of the known relevant documents, and there are fewer significant differences. But the same overall trends are apparent, confirming our findings.

## 4 Conclusions

In the biomedical domain with its complex information needs, it turns out that Boolean querying is on a par with ranked approaches using TRC queries built up from review title, research question, and inclusion criteria. Extended Boolean retrieval models are able to increase the fraction of relevant documents found after inspecting the usual 500 to 2,000 documents, by loosening the strictness of conjunctive operators and introducing some elements of ranking. This flexibility allows users to consciously choose the investment they are willing to make in inspecting answers. In our experiments, a simple extended Boolean retrieval model with binary term weights outperformed pure Boolean and ranked retrieval. Finding more documents early in the search process reduces the risk of biasing the outcome of the later employed discovery techniques, such as following citation links of already found publications or asking their authors.

If Boolean result sets are required, the proposed approach can be combined with strict Boolean querying at least in the following two approaches. First, ordering Boolean result sets by extended Boolean scores allows assessment of the quality of the query in support of iterative query refinement without sampling or inspecting the whole set. Second, after the Boolean set is reported, it can be extended by any number of documents. Either as-yet reported documents are returned by descending similarity score, or successively additional sets can be returned that do not match on (say) one or two conjuncts, or conjuncts that have the least impact on overall retrieval score. As is also the case with Boolean retrieval, binary term weights have the advantage that only the document itself determines its similarity score, not being dependent on properties of all documents in the collection. This is helpful to reproduce results and independent determination why a particular document has not been found with the published search strategy.

## 5 Future Work

Specific query parts should, by definition, be executed in a strict Boolean sense because they reflect inclusion criteria and are often based on (presumably) more reliable meta-data rather than free-text. More generally, this reduces to using different  $p$ -norms for different query operator types as well as individual query operators. Further improvement might also be possible if adapted ranking functions could be found for each of the used operators. For example, we are currently ignoring information by treating phrases as conjunctions.

Continuous or diversified scores allow the estimation of a cut-off level that could be used to filter newly published documents for their relevancy. Extended Boolean retrieval could be used as an alternative to a strict Boolean filter. However, the weighting scheme that we applied to document terms has been very basic and was demonstrated to be inferior to binary weights. Further improvements are likely if better weights can be assigned to document and query terms, for instance, conditioned on the results of other query parts. Also, our keyword queries are likely to still be suboptimal, but it is not clear how to get to better queries in this context.

Finally, ranking complexity is generally linear in the size of the query and the number of documents matching at least one term. The longer the query the more likely this becomes to be the whole collection. While many optimizations have been applied to ranking of keyword queries, we are unaware of efficient implementations for extended Boolean retrieval models, and plan to explore this issue as the next step in this project.

**Acknowledgements** National ICT Australia (NICTA) is funded by the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

## References

- Alison Avenell, Helen H. Handoll, and Adrian M. Grant. Lessons for search strategies from a systematic review, in The Cochrane Library, of nutritional supplementation trials in patients after hip fracture. *The American Journal of Clinical Nutrition*, 73(3):505–510, March 2001. PMID: 11237924.
- Chris C. Beahler, Jennifer J. Sundheim, and Naomi I. Trapp. Information Retrieval in systematic reviews: Challenges in the public health arena. *American Journal of Preventive Medicine*, 18(4 Suppl):6–10, May 2000. PMID: 10793275.
- Michael Bendersky and W. Bruce Croft. Discovering key concepts in verbose queries. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 491–498, Singapore, July 2008.
- Aaron M. Cohen, William R. Hersh, K. Peterson, and Po-Yin Yen. Reducing workload in systematic review preparation using automated citation classification. *Journal of the American Medical Informatics Association*, 13(2):206–219, March 2006. PMID: 16357352.
- Frank Davidoff, Brian Haynes, Dave Sackett, and Richard Smith. Evidence based medicine. *British Medical Journal*, 310(6987):1085–1086, April 1995.
- Kay Dickersin, Roberta Scherer, and Carol Lefebvre. Systematic Reviews: Identifying relevant studies for systematic reviews. *British Medical Journal*, 309(6964):1286–1291, November 1994. PMID: 7718048.

- Valery I. Frants, Jacob Shapiro, Vladimir G. Voiskunskii, and Isak Taksa. Boolean search: Current state and perspectives. *Journal of the American Society for Information Science*, 50(1):86–95, January 1999.
- William Hersh. *Information Retrieval: A Health and Biomedical Perspective*. Springer, 3rd edition, November 2008.
- William R. Hersh, Ravi Teja Bhupatiraju, Laura Ross, Aaron M. Cohen, Dale Kraemer, and Phoebe Johnson. TREC 2004 Genomics Track overview. In Ellen M. Voorhees and Lori P. Buckland, editors, *Proceedings of the 13th Text REtrieval Conference (TREC 2004)*, pages 16–19, Gaithersburg, Maryland, USA, November 2004. NIST. Special Publication 500-261.
- J. P. T. Higgins and S. Green, editors. *Cochrane Handbook for Systematic Reviews of Interventions*. Version 5.0.2 [updated September 2009]. The Cochrane Collaboration, 2008. Available from <http://www.cochrane-handbook.org>.
- Sarvnaz Karimi, Justin Zobel, Stefan Pohl, and Falk Scholer. The challenge of high recall in biomedical systematic search. In *ACM 3rd International Workshop of Data and Text Mining Methods in Bioinformatics (DTMBIO '09)*, November 2009.
- Joon Ho Lee. Properties of extended Boolean models in Information Retrieval. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 182–190, Dublin, Ireland, 1994. Springer-Verlag New York, Inc.
- Joon Ho Lee. Analyzing the effectiveness of extended Boolean models in Information Retrieval. Technical Report TR95-1501, Cornell University, 1995.
- David Martinez, Sarvnaz Karimi, Lawrence Cavedon, and Timothy Baldwin. Facilitating biomedical systematic reviews using ranked text retrieval and classification. In *Proceedings of the 13th Australasian Document Computing Symposium (ADCS '08)*, pages 3–10, Hobart, Tasmania, Australia, December 2008.
- Jessie McGowan and Margaret Sampson. Systematic reviews need systematic searchers. *Journal of the Medical Library Association*, 93(1):74–80, January 2005. PMID: 15685278.
- Alistair Moffat and Justin Zobel. Rank-biased precision for measurement of retrieval effectiveness. *ACM Transactions on Information Systems*, 27(1):1–27, December 2008.
- Chris D. Paice. Soft evaluation of Boolean search queries in Information Retrieval systems. *Information Technology: Research and Development*, 3(1):33–41, January 1984.
- Tadeusz Radecki. Fuzzy set theoretical approach to document retrieval. *Information Processing & Management*, 15(5):247–259, 1979.
- Stephen E. Robertson, Steve Walker, Micheline Hancock-Beaulieu, Mike Gattford, and Alison Payne. Okapi at TREC-4. In *The 4th Text REtrieval Conference (TREC-4)*, pages 73–96, Gaithersburg, Maryland, USA, November 1995. NIST. Special Publication 500-236.
- Gerard Salton, Edward A. Fox, and Harry Wu. Extended Boolean Information Retrieval. *Communications of the ACM*, 26(11):1022–1036, November 1983.
- Margaret Sampson and Jessie McGowan. Errors in search strategies were identified by type and frequency. *Journal of Clinical Epidemiology*, 59(10):1057.e1–1057.e9, October 2006. PMID: 16980145.
- Margaret Sampson, Jessie McGowan, Jennifer Tetzlaff, Elise Cogo, and David Moher. No consensus exists on search reporting methods for systematic reviews. *Journal of Clinical Epidemiology*, 61(8):748–754, August 2008. PMID: 18586178.
- Kaveh G. Shojania, Margaret Sampson, Mohammed T. Ansari, Jun Ji, Steve Doucette, and David Moher. How quickly do systematic reviews go out of date? A survival analysis. *Annals of Internal Medicine*, 147(4): 224–233, August 2007. PMID: 17638714.
- Maria E. Smith. *Aspects of the P-Norm model of Information Retrieval: Syntactic query generation, efficiency, and theoretical properties*. PhD thesis, Cornell University, May 1990.
- W. G. Waller and Donald H. Kraft. A mathematical model of a weighted Boolean retrieval system. *Information Processing and Management*, 15(5):235–245, 1979.
- Li Zhang, Isola Ajiferuke, and Margaret Sampson. Optimizing search strategies to identify randomized controlled trials in MEDLINE. *BMC Medical Research Methodology*, 6(1):23, May 2006. PMID: 16684359.
- Justin Zobel, Alistair Moffat, and Laurence Park. Against recall: Is it persistence, cardinality, density, coverage, or totality? *SIGIR Forum*, 43(1):3–15, June 2009.



# Average Distance as a Predictor of Synchronisability in Networks of Coupled Oscillators

Anthony H. Dekker

Defence Science and Technology Organisation (DSTO)  
Defence Establishment Fairbairn  
Department of Defence  
Canberra, ACT, 2600, Australia

dekker@acm.org

## Abstract

The importance of networks of coupled oscillators is widely recognized. Such networks occur in biological systems like the heart, in chemical systems, in computational problems, and in engineering systems. Systems of coupled oscillators can also be used as an abstract model for synchronisation in organisations. Here we show that synchronisability in a specific coupled-oscillator model, the Kuramoto model, is best predicted using the average distance (or characteristic path length) between nodes in the network. We do this by simulating the Kuramoto dynamics on a collection of networks of varying type, including Random, Small-World, and Scale-Free networks. Furthermore, we show that, for several real-world networks, a simple estimate based on the average distance can predict the coupling required for networks to synchronise within a threshold time.

**Keywords:** Kuramoto model, coupled oscillators, synchronisation, network, average distance.

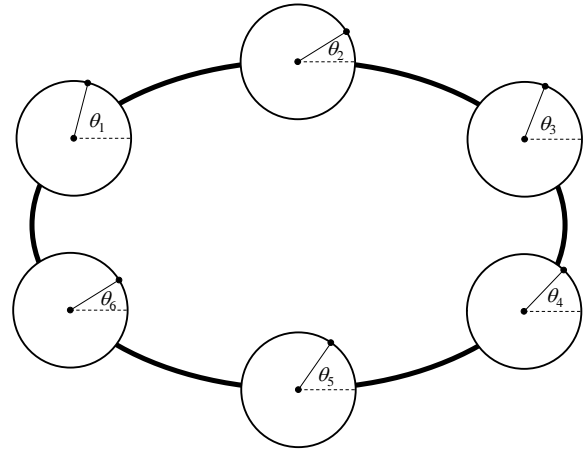
## 1 Introduction

The importance of networks of coupled oscillators is widely recognized (Strogatz, 2003). Such networks occur in biological systems like the heart (Winfree, 1980), in chemical systems (Kuramoto, 1948), in computational problems (Lee and Lister, 2008), and in engineering systems (Olfati-Saber *et al.*, 2007). Systems of coupled oscillators can also be used as an abstract model for synchronisation in organisations (Dekker, 2007a; Kalloniatis, 2008).

The Kuramoto model (Strogatz, 2000; Dorogovtsev *et al.*, 2008) is a simple system of  $n$  coupled oscillators  $O_1 \dots O_n$ , each with a natural frequency  $f_i$  (assumed to come from a unimodal and symmetric distribution) and a phase angle  $\theta_i$  (Figure 1 provides an example). Phase angles change so as to become closer to those of neighbouring oscillators, according to the differential equation:

$$\theta'_i = f_i + k \sum_j^{i \leftrightarrow j} \sin(\theta_j - \theta_i) \quad (1)$$

where the sum is taken over all oscillators  $O_j$  connected to  $O_i$  in the network (this equation is sometimes also written with the coupling constant  $k$  scaled by a factor of  $n$ ).



**Figure 1:** Six Kuramoto oscillators connected in a ring network, showing the phase angle  $\theta_i$  of each oscillator.

The degree of synchronisation in the network is measured by the correlation  $r$  between the phases of all the oscillators (Strogatz, 2000), which is equal to 1 when all the phases are identical:

$$r = \sqrt{\left(\frac{\sum_i \sin \theta_i}{n}\right)^2 + \left(\frac{\sum_i \cos \theta_i}{n}\right)^2} \quad (2)$$

In the case of a completely connected network, Kuramoto showed that the oscillators began to synchronise as the coupling constant  $k$  in Equation (1) exceeded a critical threshold (Strogatz, 2000):

$$k_c = \frac{2}{ng(f_0)\pi} \quad (3)$$

where  $g()$  is the probability density function for frequencies, and  $f_0$  is the centre or peak of the distribution. In the case that the frequencies are taken from a uniform distribution of width  $w$  (i.e. uniformly from  $f_0 - w/2 \dots f_0 + w/2$ ), Equation (3) reduces to:

$$k_c = \frac{2w}{n\pi} \quad (4)$$

For more general networks, analytical results are limited (Strogatz, 2000). Nevertheless, a simple optimistic (low) estimate for the critical coupling can be made that assumes that purely local coupling is sufficient

for synchronisation. This estimate simply scales Equation (4) by  $n/d$ , where  $d$  is the average degree (average number of links per node):

$$k_L = \frac{2w}{d\pi} \quad (5)$$

A more pessimistic (higher) estimate can be made, noting that every pair of oscillators is in fact connected, not necessarily by a link, but by a path of average distance  $D$ . Since the coupling attenuates by a factor of  $k$  for each link in the path, this yields:

$$k_H^D = \frac{2w}{n\pi} \quad (6)$$

or:

$$k_H = \sqrt[p]{\frac{2w}{n\pi}} \quad (7)$$

For large networks, this equation yields reasonable values for  $k_H$  only when average distances in the network are small, in the sense of scaling logarithmically with  $n$ . This is consistent with the importance of the “small world” effect identified by Watts and Strogatz (1998).

Jadbabaie *et al.* (2004) conducted a stability analysis on Equation (1) and derived a lower bound on the critical coupling of:

$$k_J = \frac{2\|f\|_2}{\lambda_2 \sqrt{n}} \quad (8)$$

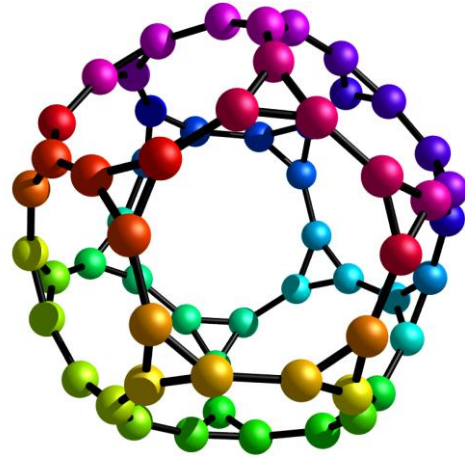
where  $\|f\|_2$  is the 2-norm of the frequency distribution, and  $\lambda_2$  is the so-called *algebraic connectivity*: the smallest non-zero eigenvalue of the Laplacian matrix for the network (the use of the Laplacian matrix arises naturally in stability analysis).

It must be noted that, for some networks, synchronisation is not guaranteed even for very high couplings  $k$ . In particular, ring-like networks have stable locally synchronised solutions (Winfrey, 1980) where, for small integers  $m$ :

$$\theta_i = \theta_0 + \frac{2mi\pi}{n} \quad (9)$$

So that, for example,  $\sin(\theta_{i-1} - \theta_i) + \sin(\theta_{i+1} - \theta_i) = 0$ . The system may fail to synchronise globally by becoming “trapped” in such a locally synchronised state. Similar locally synchronised solutions occur for spherical networks, such as the one shown in Figure 2.

Although the stability of the globally synchronised state is important, it is equally important to consider the *synchronisability* of the system – the ease with which the globally synchronised state is achieved (if at all). As a measure of the synchronisability, we take the median time  $t$  to reach a correlation of  $r = 0.9$  (in our simulations of the Kuramoto model, all systems having reached this point continue to have  $r$  asymptotically approach 1). We calculate the median over 101 simulation runs with coupling constant  $k = 0.1$  and with different (randomly chosen) initial phase angles  $\theta_i$  and frequencies  $f_i$  (the latter from a uniform distribution of width  $w = 0.001$ ). The use of the median allows for some runs becoming “trapped” in locally synchronised states.

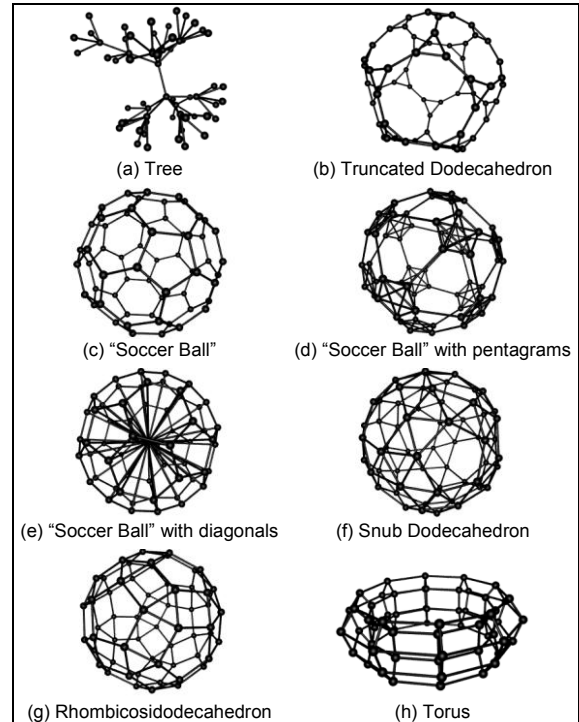


**Figure 2:** Locally synchronised state for a spherical (truncated dodecahedron) network. Phase angles for each oscillator are shown using colours on a colour wheel.

## 2 Simulation Experiments

In our simulation experiment, we discretised Equation (1) using 100 steps per time unit, with a varied sample of 79 networks, all with  $n = 60$  oscillators:

- 40 Random (Erdős-Rényi) networks (Bollobás, 2001), with average degrees  $d$  ranging from 3 to 10;
- 20 Scale-Free (preferential-attachment) networks (Albert and Barabási, 2002; Barabási, 2002), with average degrees  $d$  ranging from 2 to 5;
- 10 Small-World networks generated by the Watts rewiring process (Watts and Strogatz, 1998; Watts, 2003), with a probability  $p = 0.1$  of rewiring a link;
- 1 social network resulting from a survey of informal communication within an organisation (Dekker, 2007b), with average degree  $d = 4$ ; and
- the 8 networks in Figure 3, including 1 tree, 1 torus, and 6 spherical networks.



**Figure 3:** Eight of the sample networks used.



For this list of networks,  $k_L \leq 0.0003$ ,  $k_J \leq 0.003$ , and  $0.003 \leq k_H \leq 0.12$ . The selected coupling constant  $k = 0.1$  would therefore be expected to result in synchronisation difficulties for some of the networks, and hence is a good value for exploring synchronisability.

Figure 4 shows the result of the simulation experiment. The average distance  $D$  in the network turns out to be an excellent predictor of synchronisability. Linear regression on log-transformed data gives a best-fit power law:

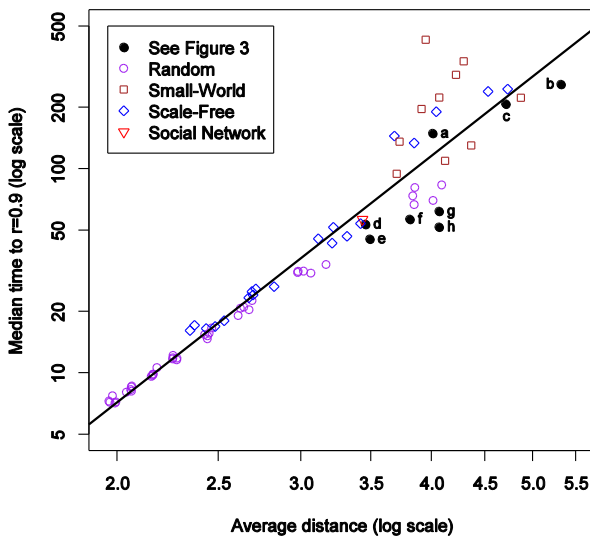
$$t = 0.44D^{4.0} \quad (10)$$

with  $R^2 = 93\%$ . In contrast, the algebraic connectivity  $\lambda_2$  is less effective as a predictor ( $R^2 = 84\%$ ), and similarly for the average degree  $d$  ( $R^2 = 68\%$ ) and the clustering coefficient (Watts and Strogatz, 1998) ( $R^2 = 3\%$ ).

Towards the top of Figure 4, the fit to Equation (10) becomes less good, and systematic differences appear between different types of network. The tree in Figure 3(a), the Scale-Free networks of average degree 2, and the Small-World networks take about 110 time units longer to synchronise than the other networks (statistically significant at the 0.001 level, by analysis of variance). There are two reasons for this difference. On the one hand, the Scale-Free networks of average degree 2 are essentially just trees and, like the tree in Figure 3(a), simply do not have enough links to synchronise well. The Small-World networks, on the other hand, retain enough of the original ring-like structure to permit distorted versions of the locally synchronised solution, which interferes with global synchronisation. In both cases, the result is poorer performance than Equation (10) would predict.

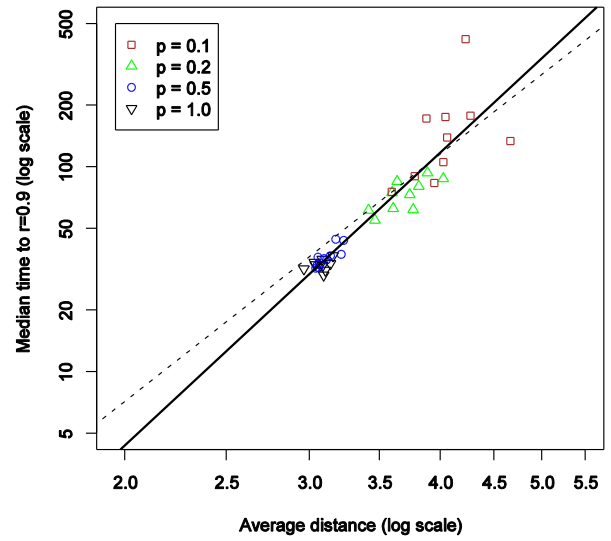
The networks which synchronise most effectively are the Random (Erdős-Rényi) networks of high degree. However, if we assign a cost to links, by minimising  $d\sqrt{t}$ , then the most efficiently synchronising networks are the Scale-Free networks with average degree  $d \geq 3$ .

**Prediction of synchronisation for some 60-node networks**



**Figure 4:** Experimental results for the set of 79 networks. The median time  $t$  to  $r = 0.9$  can be predicted quite well by  $0.44 D^{4.0}$ .

**Prediction of synchronisation for Watts-rewired networks**



**Figure 5:** Experimental results for 40 Watts-rewired networks. The best-fit power law is  $0.16 D^{4.8}$ , somewhat steeper than the power law of Figure 4 (which is shown dashed here for comparison).

The fact that Scale-Free networks with average degree  $d \geq 3$  are the most efficient synchronisers may help to explain the prevalence of Scale-Free networks in biological structures (Albert and Barabási, 2002).

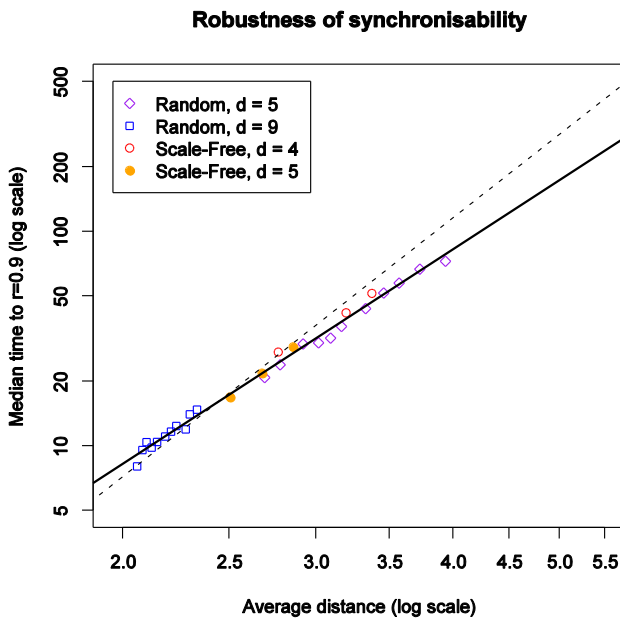
In a second experiment, we explored the synchronisation of Watts-rewired networks in more detail, using 40 networks generated by the Watts rewiring process (Watts and Strogatz, 1998; Watts, 2003), with the probability  $p$  of rewiring a link ranging from 0.1 to 1.0. Linear regression on log-transformed data gives a best-fit power law:

$$t = 0.16D^{4.8} \quad (11)$$

with  $R^2 = 85\%$ . As before, the algebraic connectivity  $\lambda_2$  is less effective as a predictor ( $R^2 = 54\%$ ). As the rewiring reduces the average distance and destroys the original ring structure, synchronisability continuously improves, at least until  $p = 0.5$ , at which point the Watts process produces networks very similar to Erdős-Rényi random networks. Figure 5 illustrates the results.

In a third experiment, we explored the robustness of synchronisability using two 60-node Random networks (with average degree  $d = 5$  and  $d = 7$  respectively, and two Scale-Free networks (with  $d = 4$  and  $d = 5$ ). We repeatedly removed the most “central” node, using the definition of centrality in Dekker (2005), and recalculated the median time to  $r = 0.9$ . As usual, the Scale-Free networks were less resistant to such targeted attacks (Albert and Barabási, 2002; Dekker and Colbert 2004), becoming disconnected after just three node removals, while the Random networks could absorb at least ten node removals. As Figure 6 shows, the time to synchronisation is predicted extremely well by the average distance ( $R^2 = 99\%$ ), with a power law:

$$t = 0.82D^{3.3} \quad (12)$$

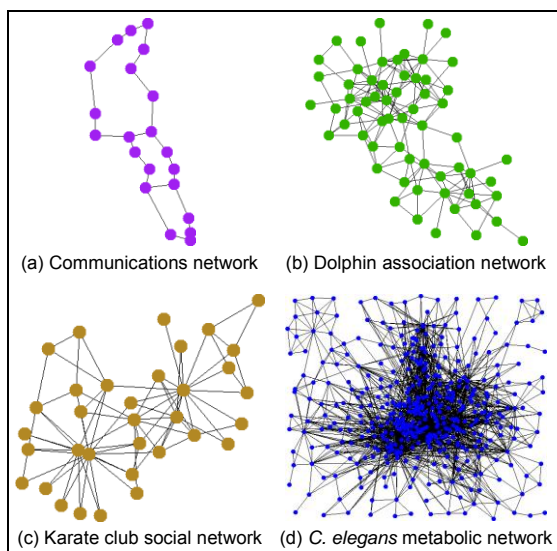


**Figure 6:** Experimental results for node removals. The best-fit power law is  $0.82 D^{3.3}$ , somewhat shallower than the power law of Figure 4 (which is shown dashed here for comparison). The spacing of the data points shows that, for the Scale-Free networks, each node removal has a larger effect on average distance (and hence on synchronisability) than for the Random networks.

The spacing of the data points in Figure 6 shows that, for the Scale-Free networks, each node removal has a larger effect on average distance and hence on synchronisability (on average, an increase of 8.6% for the Scale-Free networks per node removal, compared to 2.6% for the Random networks, a difference significant at the 0.0001 level). Scale-Free networks are therefore efficient, but not robust, synchronisers.

In a final experiment, we considered synchronisability of six real-world networks:

- a communications network for the southeast USA (Dodge, 2004), with  $n = 21$  and  $D = 4.33$ ;
- a connected subset of a scientific coauthorship network (Newman, 2006), with  $n = 57$  and  $D = 3.66$ ;



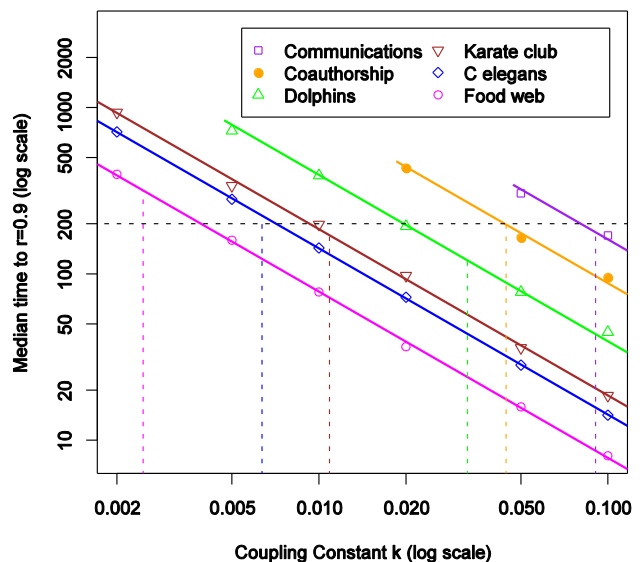
**Figure 7:** Four of the real-world networks used in the fourth experiment.

- an association network between dolphins in a community living off Doubtful Sound, New Zealand (Lusseau *et al.*, 2003), with  $n = 62$  and  $D = 3.36$ ;
- a social network from a karate club at a US university (Zachary, 1977), with  $n = 34$  and  $D = 2.41$ ;
- the (Scale-Free) metabolic network of the nematode *Caenorhabditis elegans* (Duch and Arenas, 2005), with  $n = 453$  and  $D = 2.66$ ; and
- a food web for Chesapeake Bay (Baird and Ulanowicz, 1989) considered as an undirected network, with  $n = 39$  and  $D = 1.84$ .

Figure 7 illustrates four of these networks.

For a frequency distribution width  $w = 0.001$  and a varying coupling constant  $k$ , Figure 8 shows that the median time  $t$  to  $r = 0.9$  was inversely proportional to  $k$  in each case, as would be expected. Furthermore, the pessimistic critical coupling estimate  $k_H$  (7) was in each case a reasonable predictor of the coupling needed to achieve a threshold synchronisation time of  $t = 200$  (with  $R^2 = 94\%$  for the match between prediction and actuality). This estimate therefore provides a useful guideline for assessing the synchronisability of networks.

#### Synchronisability in real-world networks



**Figure 8:** Experimental results for third experiment. The median time  $t$  to  $r = 0.9$  is inversely proportional to the coupling constant  $k$ , and the pessimistic critical coupling estimate  $k_H$  (dashed lines) is in each case a reasonable predictor of the coupling needed to achieve  $t = 200$ .

Missing data points represent values of  $k$  for which synchronisation did not occur within a limit of 1000 time units.

### 3 Discussion

The simulation experiments reported here have implications for the design of any networked system which is expected to synchronise – in particular, for the design of organisational structures (Dekker, 2007a), multi-agent systems (Olfati-Saber *et al.*, 2007), and computer networks.

Our results underscore the importance of a low average distance for the underlying network topology – the “small world” condition (Watts and Strogatz, 1998).

However, the importance of a low average distance raises the question: how low is low enough? The pessimistic critical coupling estimate  $k_H$  which we have introduced here provides an indication of whether the average distance in a network is sufficiently low. Substituting  $w = 0.001$  in equation (7) for the networks in our fourth experiment gives values of  $k_H$  in the range 0.0025 to 0.091. For comparison, the network for the Western States Power Grid studied by Watts and Strogatz (1998) has  $n = 4,941$  and  $D = 18.7$ , giving  $k_H = 0.43$ . This rather high value suggests that the power grid could be subject to synchronisation difficulties – so that the 1996 West Coast power outages (Neumann, 1996) are not altogether surprising.

Scale-Free networks with average degree  $d \geq 3$  are efficient synchronisers, which may help to explain the prevalence of Scale-Free networks in biological structures (Albert and Barabási, 2002). However, the synchronisability of Scale-Free networks is not robust against the targeted removal of nodes.

#### 4 References

- Albert, R. and Barabási, A.-L. (2002), “Statistical mechanics of complex networks,” *Reviews of Modern Physics*, **74**, 47–97.
- Baird, D. and Ulanowicz, R.E. (1989), “The seasonal dynamics of the Chesapeake Bay ecosystem,” *Ecol. Monogr.*, **59**, 329–364.
- Barabási, A.-L. (2002), *Linked: The New Science of Networks*, Perseus Publishing, Cambridge, MA.
- Ben-Naim, E., Frauenfelder, H., and Toroczkai, Z (2004), *Complex Networks*, Springer, Berlin.
- Bollobás, B. (2001), *Random Graphs*, 2<sup>nd</sup> edition Cambridge University Press, Cambridge, UK.
- Dekker, A.H. and Colbert, B. (2004), “Scale-Free Networks and Robustness of Critical Infrastructure Networks,” *Proceedings of the 7th Asia-Pacific Conference on Complex Systems*, Cairns, December, 685–699: [www.complexity.org.au/ci/vol12/msid04/](http://www.complexity.org.au/ci/vol12/msid04/)
- Dekker, A.H. (2005), “Conceptual Distance in Social Network Analysis,” *Journal of Social Structure*, **6**(3): [www.cmu.edu/joss/content/articles/volume6/dekker/](http://www.cmu.edu/joss/content/articles/volume6/dekker/)
- Dekker, A.H. (2007a), “Studying Organisational Topology with Simple Computational Models,” *Journal of Artificial Societies and Social Simulation*, **10**(4): [jasss.soc.surrey.ac.uk/10/4/6.html](http://jasss.soc.surrey.ac.uk/10/4/6.html)
- Dekker, A.H. (2007b), “Realistic Social Networks for Simulation using Network Rewiring,” *MODSIM International Congress on Modelling and Simulation*, Oxley, L. and Kulasiri, D. (eds), 677–683: [www.mssanz.org.au/MODSIM07/papers/13\\_s20/RealisticSocial\\_s20\\_Dekker\\_.pdf](http://www.mssanz.org.au/MODSIM07/papers/13_s20/RealisticSocial_s20_Dekker_.pdf)
- Dodge, M., (2004), “An Atlas of Cyberspaces,” [www.cybergeography.org/atlas/more\\_isp\\_maps.html](http://www.cybergeography.org/atlas/more_isp_maps.html)
- Dorogovtsev S.N., Goltsev, A.V., and Mendes, J.F.F. (2008), “Critical phenomena in complex networks,” *Reviews of Modern Physics*, **80**(4): arXiv:0705.0010v2
- Duch, J. and Arenas, A. (2005), “Community identification using Extremal Optimization,” *Physical Review E*, **72**, 027104.
- Jadbabaie, A., Motee, N., and Barahona, M. (2004), “On the Stability of the Kuramoto Model of Coupled Nonlinear Oscillators,” *American Control Conference*.
- Kalloniatis, A. (2008), “From Kuramoto to Boyd: applying networked dynamical systems to military Command and Control,” *International Workshop on Complex Systems and Networks*, Canberra: [labnetcon.anu.edu.au/IWCSN/Lecture\\_Slides/2008-10-03-Kalloniatis.pdf](http://labnetcon.anu.edu.au/IWCSN/Lecture_Slides/2008-10-03-Kalloniatis.pdf)
- Kuramoto, Y. (1948), *Chemical Oscillations, Waves, and Turbulence*, Springer, Berlin.
- Lee, S. and Lister, R. (2008), “Experiments in the Dynamics of Phase Coupled Oscillators When Applied to Graph Colouring,” 31<sup>st</sup> Australasian Computer Science Conference (ACSC), Wollongong, Dobbie, G. and Mans, B. (eds), *Conferences in Research and Practice in Information Technology*, **74**, 83–89: [crpit.com/confpapers/CRPITV74Lee.pdf](http://crpit.com/confpapers/CRPITV74Lee.pdf)
- Lusseau, D., Schneider, K., Boisseau, O.J., Haase, P., Slooten, E., and Dawson, S.M. (2003), “The bottlenose dolphin community of Doubtful Sound features a large proportion of long-lasting associations,” *Behavioral Ecology and Sociobiology*, **54**, 396–405.
- Neumann, P.G. (1996), “Western U.S. power blackout,” *Risks Digest*, **18**(25): [catless.ncl.ac.uk/risks/18.25.html](http://catless.ncl.ac.uk/risks/18.25.html)
- Newman, M.E.J. (2006), “Finding community structure in networks using the eigenvectors of matrices,” *Phys. Rev. E*, **74**, 036104.
- Olfati-Saber, R., Fax, J.A., and Murray, R.M. (2007), “Consensus and Cooperation in Networked Multi-Agent Systems,” *Proceedings of the IEEE*, **95**(1), 215–233.
- Strogatz, S.H. (2000), “From Kuramoto to Crawford: exploring the onset of synchronization in populations of coupled oscillators,” *Physica D: Nonlinear Phenomena*, **143**, 1–20.
- Strogatz, S. (2003), *Sync: The Emerging Science of Spontaneous Order*, Hyperion, New York.
- Watts, D.J. and Strogatz, S.H. (1998), “Collective dynamics of ‘small world’ networks,” *Nature*, **393**, 440–442.
- Watts, D.J. (2003), *Six Degrees: The Science of a Connected Age*, William Heinemann, London.
- Winfree, A.T. (1980), *The Geometry of Biological Time*, Springer, New York.
- Zachary, W.W. (1977), “An information flow model for conflict and fission in small groups,” *Journal of Anthropological Research*, **33**, 452–473.



# Applying a Neural Network to Recover Missed RFID Readings

Peter Darcy

Bela Stantic

Abdul Sattar

Institute for Integrated and Intelligent Systems  
Griffith University, Queensland, Australia,  
Email: {P.Darcy, B.Stantic, A.Sattar}@griffith.edu.au

## Abstract

Since the emergence of Radio Frequency Identification technology (RFID), the community has been promised a cost effective and efficient means of identifying and tracking large sums of items with relative ease. Unfortunately, due to the unreliable nature of the passive architecture, the RFID revolution has been reduced to a fraction of its intended audience due to anomalies such as missed readings. Previous work within this field of study have focused on restoring the data at the recording phase which we believe does not allow enough evidence for consecutive missed readings to be corrected. In this study, we propose a methodology of intelligently imputing missing observations through the use of an *Artificial Neural Network* (ANN) in a static environment. Through experimentation, we discover the most effective algorithm to train the network is via genetic training with a high chromosome population. We also establish that the ANN restores a cleaner data set than other intelligent classifier methodologies in the majority of the test cases especially when faced with large amounts of missing data.

**Keywords:** Radio Frequency Identification - RFID, Artificial Neural Network, Data Cleaning.

## 1 Introduction

The alluring promise of an automatic and wireless technology that is cost effective and efficient has been the main appeal for researchers to find a feasible way to implement the passive radio frequency identification architecture. Unfortunately, there are several factors limiting such widespread acceptance of the system, one of which is the data anomalies, such as missed, wrong and duplicate readings. These anomalies are caused due to the use of RFID passive tags to conduct the process. If this were to be corrected, high laboured tasks, such as inventory checks or supermarket shopping, can benefit greatly by cutting costs in manual labour and time.

A way to enhance the observational data to allow it to be meaningful in applications is to employ an intelligent algorithm such as a classifier that can be applied to correct ambiguous entries when given several different options to enhance the system. The *Artificial Neural Network* trained using back-propagation or an evolutionary algorithm has been shown in various applications as an effective and efficient classifier

that can be utilised to discover the correct output when given various situations.

Common methodologies that attempt to correct data anomalies found in RFID observational data either use filtration applied to streamed data at the reader level or correction algorithms applied to data that has been already stored. In literature, a filtration algorithm has been mentioned as ideal for most situations that do not need analysis of the whole data set. However, this limits its cleaning potential to only a fraction of the anomalies. Also, most deferred algorithms lack intelligence which need to be applied when ambiguous situations occur. Additionally, in previous work we utilised several intelligent methods such as Bayesian Networks and Non-Monotonic Reasoning applied at a deferred stage to correct missed data anomalies. With regards to the scope of our work, we focus directly on RFID technology and previous methodologies that have been proposed to correct such errors.

In this work, we propose the use of an Artificial Neural Network (ANN) with an analysis technique to correct stored observational data from a static RFID-enabled environment. In essence, we are attempting to correct the fundamental problem of missing spatial-temporal data by replacing it with generated values based on the shortest path between the two gaps. We have examined the most effective training algorithm of two prominent ANN algorithms and found that the Genetic Algorithm training provided a thorough cleaning result. We also compared the neural network, found from the training experimentation, with two Deferred Bayesian Networks and found that the ANN provided a higher cleaning rate.

The remainder of this paper is organised as follows: In Section 2 we provide a brief survey of Radio Frequency Identification and Artificial Neural Networks. Section 3 summarises the state-of-the-art methodologies that are currently employed to correct observational data. An analysis of our methodology is provided in Section 4, which will be broken up into the analysis phase, correction phase and Neural Network of our algorithm. The exact nature of our experimentation is explained in Section 5 followed by results and analysis in Section 6. Finally, in Section 7, we conclude our work and outline future work.

## 2 Background

Since the introduction of the passive radio frequency identification (RFID) system, the scientific community has been attempting to correct the significant anomalies present in the architecture that prevent it from being widely adopted in real-world scenarios. Prevalent anomalies present in the data warehouse include observational records that are missed within the data collection cycle. To combat this, a higher

level of intelligence is needed to be applied to the data sets to impute and restore the missed data. One such method is a classifying algorithm which would be utilised to discover the correct missing data, such as an *Artificial Neural Network* (ANN). An ANN operates by reading inputs, which it then trains upon to classify to achieve the desired output. By the end of its training, it is hoped that the ANN will be able to classify the input into a correct output.

## 2.1 Radio Frequency Identification

Radio Frequency Identification is a collection of wireless devices utilised together to recognise large sums of items. The system has already employed in several real-world applications such as postal-tracking, aviation-parts identification and baggage handling at airports (Collins 2004). As seen in Figure 1, there are four main components that combine to effectively create a RFID system: the reader, tag, middleware and data warehouse. The reader will search the area the user desires to record tag's identifier that responds, the time that this reading took place and the identifier of the reader. The tag will then transmit its unique identifier named the Electronic Product Code (EPC) to the reader. These recorded observations are then delivered to the middleware to perform immediate correcting techniques and store the information into a data warehouse (Chawathe et al. 2004).

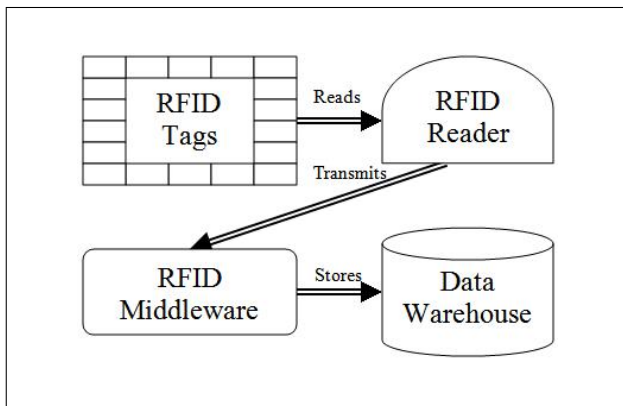


Figure 1: A high level diagram of how data is passed through the passive RFID architecture.

When employing a RFID system, there are several different tags utilised to allow items to be identified. These include passive, semi-active and active tags. The passive tag is the cheapest tag due to the lack of any battery source which also theoretically allows it to have an indefinite lifespan. It extracts its power from the electro-magnetic pulse generated from the reader and uses it to fire its unique identifier back to the reader. The semi-active tag utilises a battery as a power source but only to extend the range of the transmission. It still utilises the pulse emitted from the reader to send its identifier back which results in it having a 5-10 year lifespan. The last tag is the most advanced and expensive identifier on the market, the active tag. It utilises the battery to both extend its reading range and send its identifier. This, regrettably, results in the active tag having a lifespan of only 1-5 years long (Chawathe et al. 2004).

Unfortunately, there are severe problems that hinder the wide scale deployment of RFID systems across the commercial sector. The four main problems are identified as follows: the data generated is low level, as seen in Table 1; the data is error prone; the data generated is received in high volumes and the spatial-temporal nature of the data is complex. The data

generated from the readers are in the form of a tag identifier, timestamp and reader identifier. Unfortunately, without transforming the data to higher level information, such as events, using additional tag and reader information, the information is useless (Khousainova et al. 2007). Most RFID systems, especially the architectures that rely on passive tags, generate three core anomalies: wrong, missing and duplicate data. Due to the continuing scan of tags every second or less, the data storage suffers from the problem of massive intakes of data. It has been stated that Wal\*Marts in America which have employed the use of RFID technology generate 7TB of data daily (Raskino et al. 2005). With the increasing developments of passive technology, there is also the issue of complexity with regards to spatial and temporal data (Wang & Liu 2005). For example, if there is small hand-held reader utilised to scan different locations, the spatial location of the reader will also constantly be different.

Tag EPC	Reader ID	Timestamp
t1	r1	2009-08-15 14:05:08.002
t2	r1	2009-08-15 14:05:08.002
t1	r2	2009-08-15 14:45:54.028
t2	r1	2009-08-15 14:05:08.002
t1	r3	2009-08-15 15:02:06.029
t2	r1	2009-08-15 15:02:06.029
t1	r4	2009-08-15 15:18:49.016
t2	r1	2009-08-15 15:18:49.016

Table 1: A table populated with sample RFID Data containing the information of EPC, Reader and Timestamp. Please note that actual RFID data would contain unique identifiers for the tag EPC and reader identifier.

As mentioned above, the data anomalies within the generated data include missed, wrong and duplicate observational (Jeffery et al. 2006)(Bai et al. 2006). Without the correction of these anomalies, the passive RFID architecture will never be utilised to its full potential as a cost effective and efficient means of wireless identification of large sums of items. Of the three anomalies, missing observations may be considered the hardest to restore as the data is not recorded into the database and, therefore, does not have as much analytical information when attempting to correct the records. It has been estimated that reader only captures 60%-70% of the data that was supposed to be recorded (Floerkemeier & Lampe 2004).

## 2.2 Artificial Neural Networks

Artificial Neural Networks (ANN) is a classifying tool that is modelled after the biological neural network found in the brains of animals. There are four main components of a biological neuron in the brain as seen in Figure 2: the dendrites that receive the information; the cell body that reacts to the information given a certain threshold; the axon that is the output of the neuron and the synapses which connects various dendrites to axons. The brain itself possesses a large sum of neurons which all work in unison to form a neural network, it is estimated that in humans, the amount of neurons presently in the brain range from 10 billion to 1 trillion (Williams & Herrup 1988).

The ANN operates by receiving inputs, manipulating them through the use of weights between hidden units and hidden layers in each neuron to provide an output, as shown in Figure 3 (McCulloch & Pitts 1943). Its main advantage is that it is able to generalise information and provide a relatively correct answer. There are many ways through which to train the weights of an ANN through means of



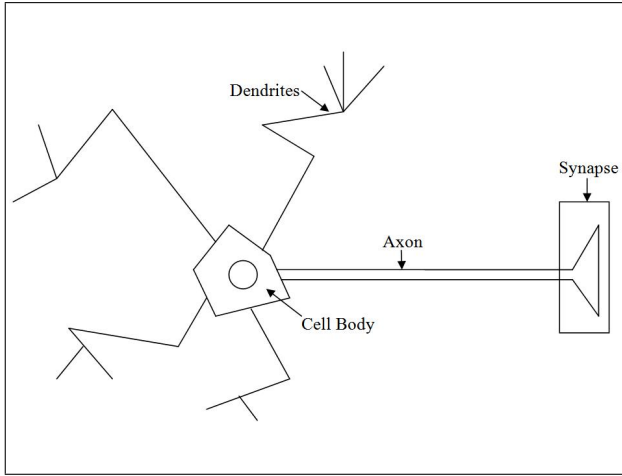


Figure 2: A diagram representing a biological neuron's structure found in living organisms.

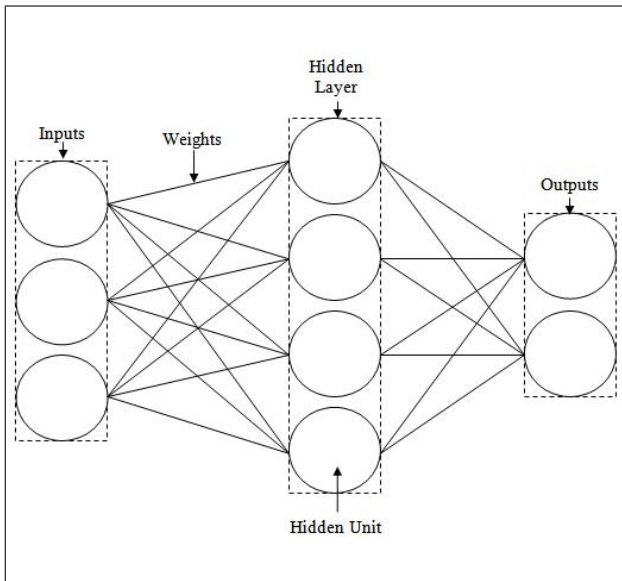


Figure 3: A high level diagram of the various components that create a feed-forward perceptron neural network.

algorithms such as Back-Propagation or Evolutionary Neural Networks. An artificial neuron pictured in Figure 4 is composed of various inputs that are manipulated by weights which are summed to find the neural activity for that specific neuron. Activation Functions are then utilised to change the output to a desirable answer that can be used in the applications. There are two current prominent activation functions: the hard limiter, which will either classify the output as positive one or below zero, and the sigmoidal function, which applies the equation found in Equation 1 to the output.

$$f(x) = 1/(1 + \exp(-x)) \quad (1)$$

### 2.2.1 Back-Propagation

The Back-Propagation algorithm utilises the overall error found in the actual outputs when compared with the desired outputs to manipulate the weights of the neural network. As seen in Figure 5, the back-propagation algorithm procedures are as follows (Rumelhart et al. 1986); the user presents the training sets to the network in the form of inputs and desired outputs; the network calculates the error rate

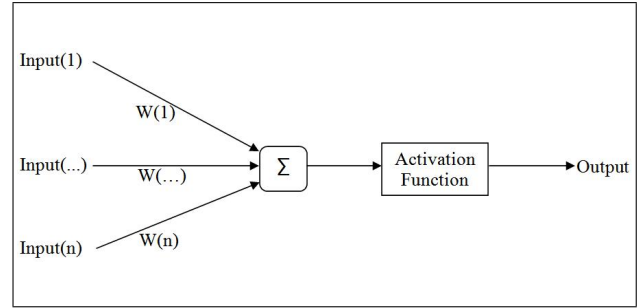


Figure 4: A diagram representing an artificial neuron's structure found in artificial neural networks.

between the actual output and desired output and manipulates accordingly to obtain a lower error rate on the next round. The algorithm is stopped with only two criteria. The first is a user specified number of iterations and the second is a Root-Mean-Square (RMS) error threshold also specified by the user (Blumenstein et al. 2007).

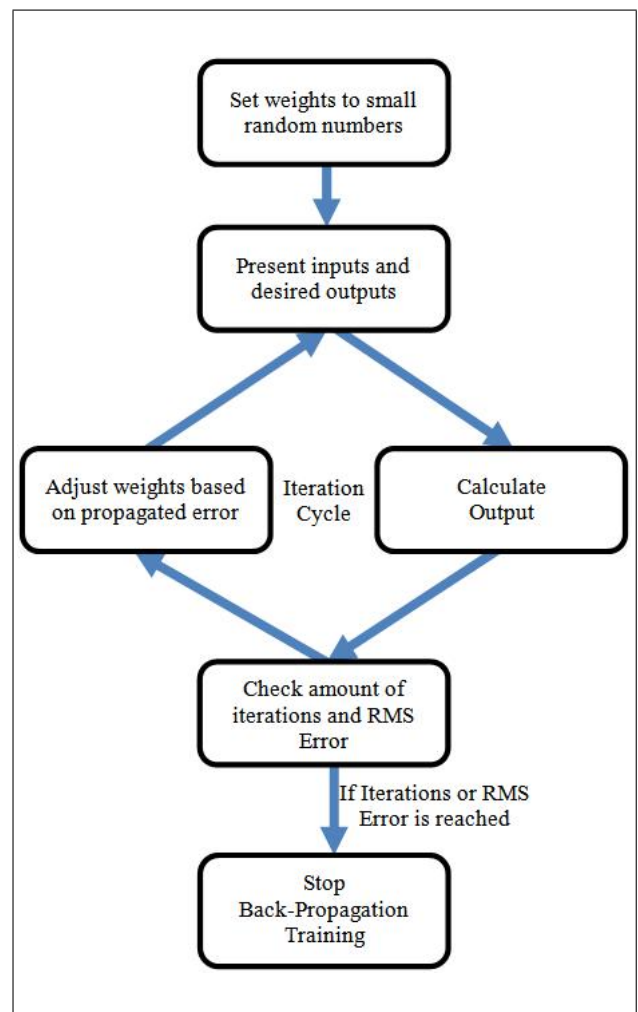


Figure 5: A high level flow diagram of the processes necessary to train a neural network using the back-propagation algorithm.

### 2.2.2 Evolutionary Neural Networks

An Evolutionary Neural Network is an ANN that utilises a genetic algorithm modelled after the theory of evolution to manipulate the weights accordingly (Holland 1975). It does this by initialising chromosomes that have information of all the weights of the



neural network to small values and finding the most correct values (Rooij et al. 1996). A certain threshold will be used to delete the remaining chromosomes and proceed to cross over the fit chromosomes to increase the population to its original size. Additionally, a mutation will occur on a small selected amount of the population to avoid local minima. The stopping criteria is a user-specified correctness threshold or number of generations for the chromosomes. The necessary processes and order in which they occur may be viewed in Figure 6 (Cha et al. 2008).

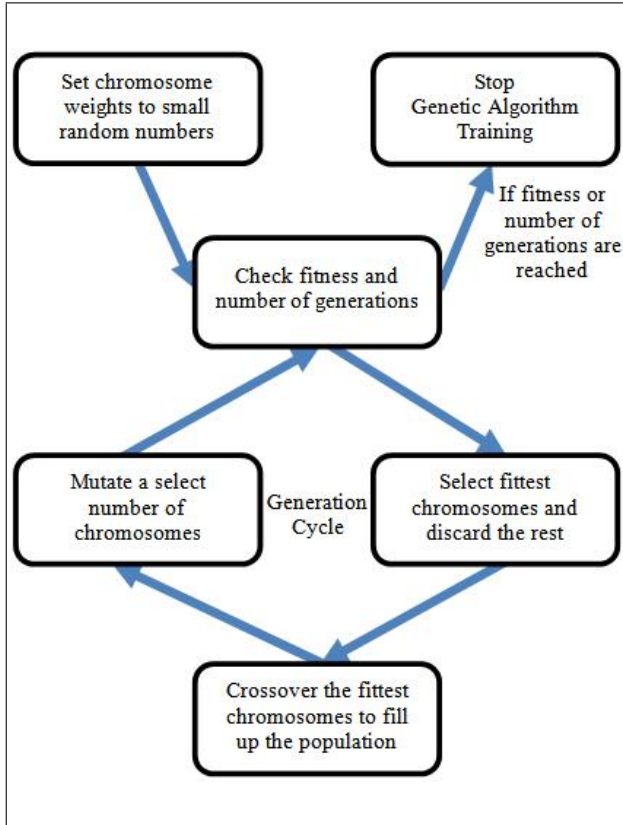


Figure 6: A high level flow diagram of the processes necessary to train a neural network using a genetic algorithm.

### 3 Related Work

The data anomaly problem of passive RFID records have been studied extensively throughout recent years with the main goal to correct the observations to allow it to be used in higher business processes. There are two main ideologies within literature. The first is to filter the incoming records when recording the data and the second is to apply correction algorithms which enhance the integrity of the observations in the stored data warehouse. Most filtration processes centre around utilising anti-collision algorithms (Shih et al. 2006) as the tags are read at the reader. Other methodologies employ rules that have been specified by the user to correct the data when it is in the database (Rao et al. 2006).

From examining the respected literature regarding these related works, we have found that the filtering and rule based methodologies sacrifice analytical knowledge and intelligence respectively which we believe is crucial to thoroughly enhance the data to its maximum potential of integrity. The filtration methods are applied at the edge and, therefore, lack analytical knowledge gained from the previous data entries and future data entries that may be examined within

the data warehouse after the recording phase. Additionally, the rule-based approach is specified by users which may allow for logical errors within the protocols and lack the intelligence needed to cope with ambiguous observations. In previous work, we have utilised Non-Monotonic Reasoning (Darcy et al. 2007), Static Bayesian Networks (Darcy et al. 2009b) and Evolutionary Bayesian Networks (Darcy et al. 2009a) to combat these drawbacks.

## 4 Methodology

The core functionality of our system which is shown in Figure 7 has been divided into two phases, the analysis phase and the correction phase. As the name implies, the analysis phase consists of an algorithm that sweeps through the RFID readings finding any missed readings that should be present. If said anomalies are found, the algorithm records various attributes regarding the gap of knowledge which are then passed on to the correction phase. The correction phase is where the system attempts to seek out the ideally correct combination of readings to impute the gapped knowledge of data. It is the goal of the system at this time to substitute the most correct data that the system can generate to allow for higher business processes to be conducted.

### 4.1 Analysis Phase

---

**Algorithm 1:** The algorithm for the Analysis Phase of the methodology. Its purpose is to discover the various algebraic values needed and find the inputs needed for the neural network.

---

```

foreach Tag do
  Start Counter at first Observation's
  Timestamp.
  foreach Reading do
    if Counter  $\neq$  Current Reading
    Timestamp then
      Record the current Reader (b) at the
      start of the Gap and the Reader
      before current (a).
      Go to the next recording after the
      Gap.
      Count the amount of Missed
      Readings in the Gap (n).
      Record the current Reader (c) at end
      of Gap and the Reader after the
      current (d).
      Find the Shortest Path between b
      and c then count how many
      observations are present (s).
      Record (a), (b), (n), (c), (d), (s) and
      the Shortest Path.
      Determine if (a==b), (b==c),
      (b<=c), (c==d), (n==(s-2)),
      (n<(s-2)) and (n>(s-2)). Record
      these values as the inputs for the
      Neural Network phase.
    end
  end
  end
  Pass the values to the Correction Phase and the
  analysis to the Neural Network phase.
  
```

---

The analysis phase was originally conceived from previous work in which observational data is analysed attempting to uncover several key factors that reflect the nature of the missed readings. The pseudo code version of the algorithm used to perform this phase

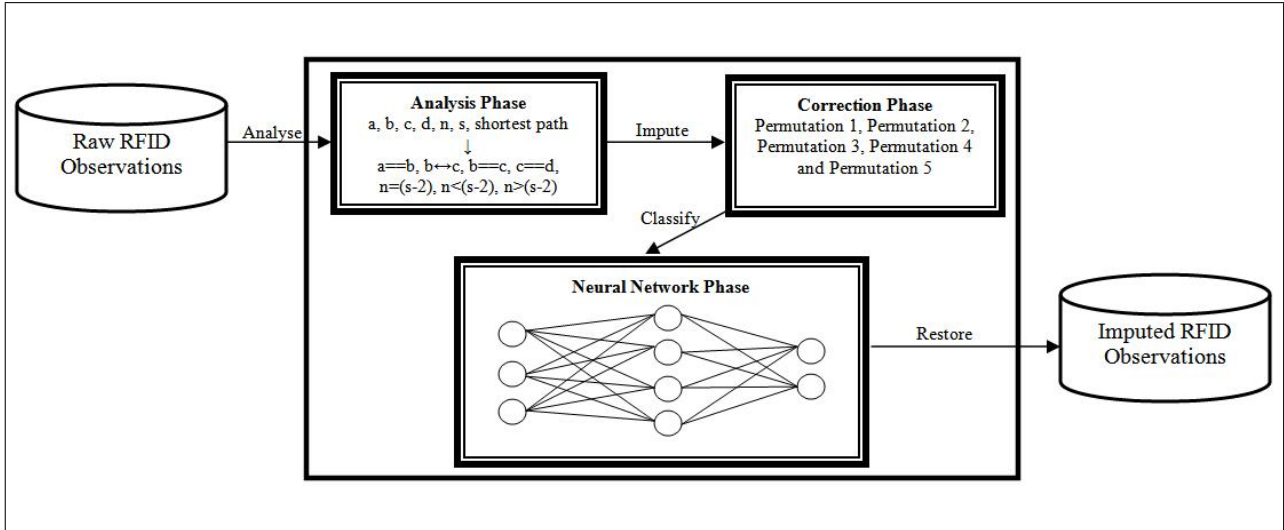


Figure 7: A high level diagram of the architecture for the methodology we have proposed which utilises intelligent analysis and a neural network to correct missed observational data.

may be found in Algorithm 1. The phase involves first separating the amalgamation of readings into individual tag streams by dividing the observations according to the identifier. After this division has been completed, we attempt to find gaps in the data that are found due to a missing periodic timestamp. We have devised our scenario with the assumption so that the readers periodically scan for readers within the vicinity of it. Additionally, it is crucial that our methodology is granted access to the map data of the readers in the static environment to impute the most likely readings.

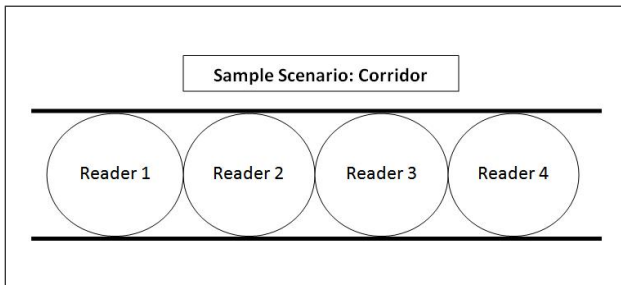


Figure 8: A sample scenario which will provide an example of how the methodology operates. The corridor environment is static with walls preventing people from not being detected from readers one to four in this order.

Once a gap of knowledge is found in the records, we analyse the surrounding readings to find six core values: the reader two observations and one observations before the gap ( $a$  and  $b$ ); the amount of the missed reading ( $n$ ); the reader one and two observations after the gap of knowledge ( $c$  and  $d$  respectively); and the minimum amount of readings that are required to join readings  $b$  and  $c$  utilising the map data ( $s$ ). Additionally, the shortest path will be required to determine the minimum amount of readers needed to bridge the gap. For example, if there is a basic map of a corridor in which readers 1-4 are connected in a linear line as shown in Figure 8, then the data analysis algorithm would detect the values as shown in Figure 9.

The analytical phase requires that the information it had just discovered be then translated into meaningful analytical information. To accomplish this, the process will find the following mathematical operations which we believe can be utilised to determine

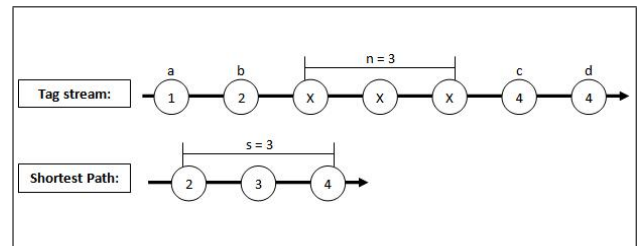


Figure 9: The variables and shortest path which are needed for the imputation process from the sample scenario which have been placed onto the sample scenario's tag stream.

the most correct readings to fill the gaps of knowledge. These analytical operations include: finding if readers " $a$ " and " $b$ " are equal ( $a==b$ ); determining if readers " $b$ " and " $c$ " are relatively close to each other one according to the map data ( $b \leftrightarrow c$ ); discovering if the readers of " $b$ " and " $c$ " are equal ( $b==c$ ); finding if readers of " $c$ " and " $d$ " are equal ( $c==d$ ) and discovering if " $n$ " is equal to, less than or greater than " $s$ " minus two ( $n==(s-2)$ ), ( $n<(s-2)$ ), ( $n>(s-2)$ ). The reasoning behind subtracting two from value " $s$ " is that the shortest path will include the values of " $b$ " and " $c$ " which are not necessarily a part of the missing gaps of knowledge. All of these analytical boolean variables are then passed on to the correction phase which utilise it to seek out the most ideal imputed reader values.

## 4.2 Correction Phase

The Correction Phase, whose pseudo code may be viewed in Algorithm 2, is where the possible permutations of the imputed data is created. It relies on the information that was obtained from the analysis processes to use as evidence to determine the most correct data. When the information is passed to this phase, five possible combinations of the data which we have termed "Permutations" will be generated. These may viewed in Figure 10. The first permutation will generate the missed readings to all have the value of reader " $b$ ". The second permutation is a mirror to the first in that it will substitute all the missed values with the reader " $c$ ". The third requires the use of the shortest path generated before to insert it into the middle of the missed observations. Any ad-

---

**Algorithm 2:** The algorithm for the Correction Phase of the methodology. Its purpose is to develop the permutations needed as the output for the neural network.

---

Receive the algebraic values of the missing data from the Analysis Phase.

**foreach** *MissingRecord* **do**

    Create an Array of size “n”.

    Generate the first permutation by using the value of “b” substituted for each recording. Generate the second permutation by using the value of “c” substituted for each recording.

    Generate the third permutation by inserting the shortest path in the middle of the array and substitute the values of “b” and “c” on the left and right side of the array to additional missing gaps.

    Generate the fourth permutation by inserting the shortest path on the right of the array and substitute the value of “b” on for all remaining gaps.

    Generate the fifth permutation by inserting the shortest path on the left of the array and substitute the value of “c” on for all remaining gaps.

**end**

Pass the permutations to the Neural Network phase.

---

ditional missed values will have reader values “b” or “c” substituted where it is in close proximity.

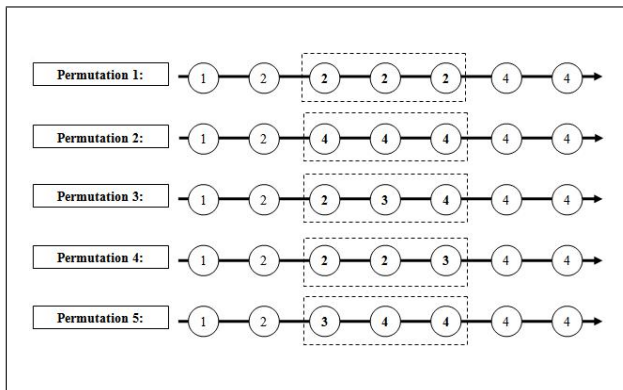


Figure 10: The permutations generated for the scenario which are to be inserted back into the data storage as imputed readings.

The fourth permutation will utilise the shortest path again, but place it on the latter missed readings rather than the middle. All missing values that are on the former side of the missing data will be substituted with the reader value “b”. The final permutation generated will once again be a mirrored duplicated of the fourth in that the shortest path will be inserted to the former part of the gap of knowledge and the reader value “c” will be substituted for any additional missed readings. We believe that these permutations represent the most likely candidates of the data in that each one finds the shortest route between the starting and destination points which we also believe is likely for the average tag. After these possible permutation combinations have been derived each one, with the addition of the extra analytical data, is then forwarded onto the ANN to determine what it believes is the most likely candidate for restoration.

### 4.3 Artificial Neural Network Phase

We have employed the use of an Artificial Neural Network (ANN) in the last phase as a means to classify which permutation is the most correct given the analysis data ascertained from the previous two phases. The ANN accepts seven binary inputs to reflect the analysis data (a==b, c==d, etc.) and has five binary outputs to reflect that the permutations that have been found. The ANN itself will be comprised of nine hidden units and one hidden layer as seen in Figure 11.

The reason why we have chosen this configuration is that we wish to have a number of units greater than the number of inputs and we believe that increasing the number of hidden layers will not necessarily increase the accuracy. We have also applied a momentum term and learning rate whose values are 0.4 and 0.6 respectively to avoid local minima and network paralysis. Each input and output value will not be 1 and 0 as this may not yield a very high classification rate, instead we will use the values of 0.9 and 0.1 respectively. We will set the stopping criteria as both the RMS error threshold when it reaches below 0.1 and 1000 iterations. We have also utilised the sigmoidal activation function to derive our outputs. The pseudo code for both the back-propagation and genetic algorithm training methods can be found Algorithms 3 and 4 respectively.

---

**Algorithm 3:** The algorithm for training the Neural Network Phase utilising the Back-Propagation Algorithm. The goal of this algorithm is to modify the weights using forward and backward passes until the network has been trained to determine the correct outputs.

---

Initialise the weights to small random values.

Present the training set to the neural network.

**foreach** *Iteration* **do**

    Determine the actual output of all the neurons (Forward Pass).

    Check stopping criteria (RMS Error and Iterations).

**if** *Stopping Criteria is True* **then**

        Break the algorithm and store the weights.

**end**

    Determine the error terms.

    Adjust the weights using the learning rate, momentum and error terms (Backward Pass).

**end**

Store highest performing network configuration.

---

## 5 Experimental Evaluation

To determine the maximum accuracy of our methodology and compare it to other state-of-the-art applications, we have devised two experiments. We have run both of these experiments on a standard computer environment at our institute to demonstrate that the system may be run in a typical environment. The two experiments have been designed to determine the best training algorithm to configure the neural network, and compare the said network to a deferred bayesian network implementation to find the significance of methodology.

### 5.1 Environment

The computational environment of our experiment consists of utilising the C++ language to write the



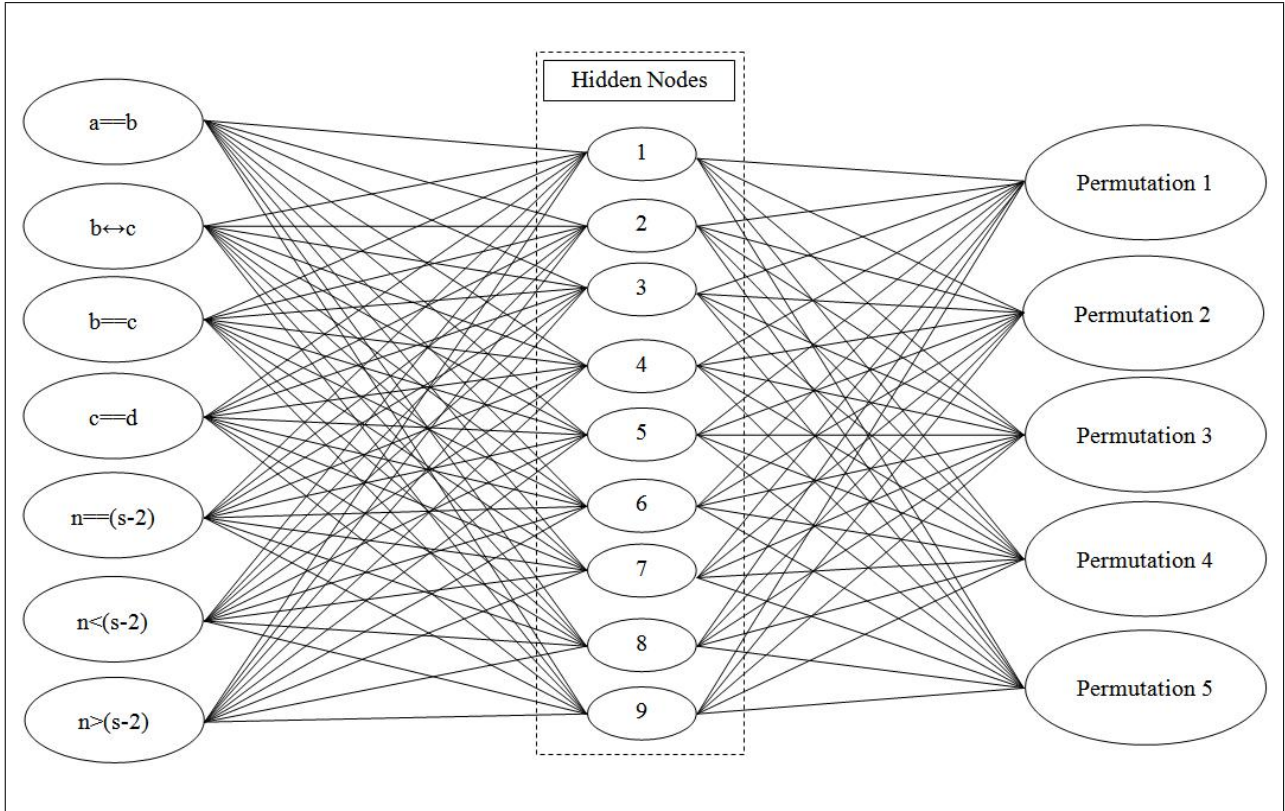


Figure 11: A high level representation describing the structure of the neural network we are utilising within our work. The 7 inputs include all the analytical information gathered from the analysis phase, whereas the 5 outputs are all five permutations which were generated during the correction phase. We have also employed 9 hidden nodes inside the network which results in the sum of 108 weights that connect each of the nodes to either the inputs or outputs.

---

**Algorithm 4:** The algorithm for training the Neural Network Phase utilising the Genetic Algorithm. The goal of this algorithm is to set the weights to a point they are able to chose the correct permutation when given various training sets.

---

```

Initialise the weights to small random values.
Present the training set to the neural network.
foreach Generation do
    Evaluate and order the chromosome
    according to fitness (the classification rate).
    Check stopping criteria.
    Check stopping criteria (Generations).
    if Stopping Criteria is True then
        Break the algorithm and store the
        weights of the highest performing
        chromosome.
    end
    Delete half the least fit population members
    and breed two random fitter chromosomes
    to fill the amount of population.
    if Child chromosome is already present in
    population then
        Discard child chromosome.
    end
    Breed two different random parents.
    end
    Apply mutation to select members of the
    chromosome by resetting the weights to
    random values - 1% the top 10% fittest
    population members, 5% for other
    chromosomes.
end
Store highest performing network configuration.

```

---

code and execute it using Microsoft Visual C++ 6.0. It has been run on a computer that has the Windows XP operating system that runs Service Pack 3 Intel (R) Pentium (R) 4 CPU 2.79 GHz with 2.00 GB of RAM.

## 5.2 Experiments

We have utilised two different types of experiments to determine the significance of our methodology. The first experiment which we labelled the “Configuration Experiment”, we have set up is designed to determine which of the training techniques yields a higher accuracy for the neural network. The two training algorithms we will compare are a back-propagation and a genetic training algorithm as we have found that they are prominent in the field of training neural networks. The second experiment which we labelled the “Significance Experiment”, we have decided to conduct involved proving the significance of the most accurate neural networks result from the configuration experiment when compared with the accuracy of a deferred bayesian network approach. We chose a deferred bayesian network as it has been shown in the past to obtain a high accuracy rate and is directly comparable to our deferred neural network approach. The training and testing sets will consist of permutation scenarios in which the worst case occurs. In reality, this scenario has an extremely low chance of occurring as our algorithm has been designed so that where the missed readings are purely random and short in size, which frequently happens in physical applications, every permutation generated will be correct.

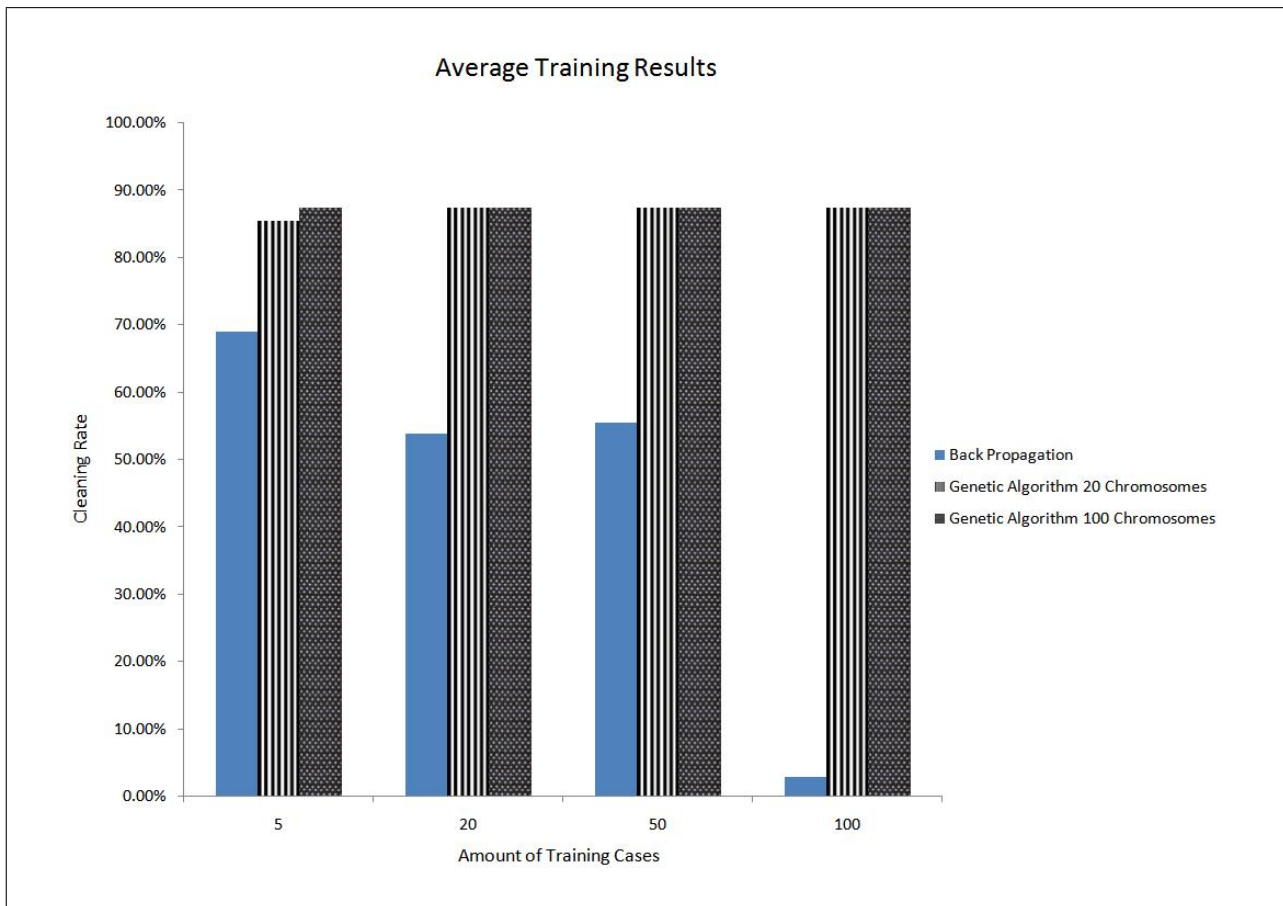


Figure 12: A bar graph depicting the averaged results of the three training algorithms versus the cleaning rate to determine the most effective network configuration for our methodology. The solid bar represents the Back-Propagation algorithm, the striped bar represents the Genetic Algorithm with 20 chromosomes applied and the dotted bar represents the Genetic Algorithm with 100 chromosomes applied.

## 6 Results and Analysis

In this work, we have proposed a methodology that harnesses a powerful data analysis technique coupled with a neural network to correct missing observations in stored RFID data. To this end, we have devised two experiments to determine the best network configuration to use when comparing prominent training algorithms and analysed the effectiveness of both it and two Bayesian Networks. We have recorded and evaluated the results of these experiments directing focus to the rate of cleaning in choosing the correct permutation to restore the observational data.

### 6.1 Configuration Experiment

The Configuration Experiment has the goal of seeking out the neural network training algorithm that yields the highest clean rate. The two training methods used for comparison are the back-propagation and genetic algorithms. The training set utilised in this experiment is comprised of every possible combination of the inputs and their respective outputs which amount to a total of 128 entries. We have conducted tests upon three different training algorithm setups, the first is the back-propagation algorithm and the other two are genetic algorithms that use 20 and 100 chromosomes to find the optimised solution. Additionally, we conducted each experiment in three trials to further generalise our results. The algorithm that had the hardest time finding the correct configuration was the back-propagation algorithm when it iterated for 50 and 100 times, earning it 1.56% classification rate. The trainer that performed the best was the

Genetic Algorithm both using 20 and 100 chromosomes in every test and iteration number excluding the 20 chromosome configuration which lasted for 5 generations.

The analysis we performed on these results consisted of us graphing the average of our findings into a bar graph to illustrate the difference in algorithms. As shown in Figure 12, on average the Neural Network Genetic Algorithm performed the best obtaining an 87.5% cleaning rate. Unfortunately, as discovered before in the results, the back-propagation algorithm performed the weakest within the three algorithms. This is especially present when it is attempting to clean after being trained for 100 iterations. We believe the poor results of the back-propagation algorithm was due directly to over-training the network. We noticed that there was also a particularly low result when attempting to train this algorithm for 50 attempts in trial 2. However, it wasn't until the 100 iteration training that we could clearly see the effects of the training routine on the average cleaning rate.

As a general observation from these averaged results, we can clearly see that, with the accepting of the 5 iterations experiment, each training result had both genetic algorithms obtaining the same cleaning rate. After examining the weights of both the 20 and 100 chromosome genetic algorithm configurations, we found that although the same result was obtained, the networks that had been generated were different. We believe that this due to there being many certain networks that may be utilised to find an average maximum of classification from the data set which happens to be 87.5% (the highest cleaning rate achieved). Furthermore, we believe that the results obtained from

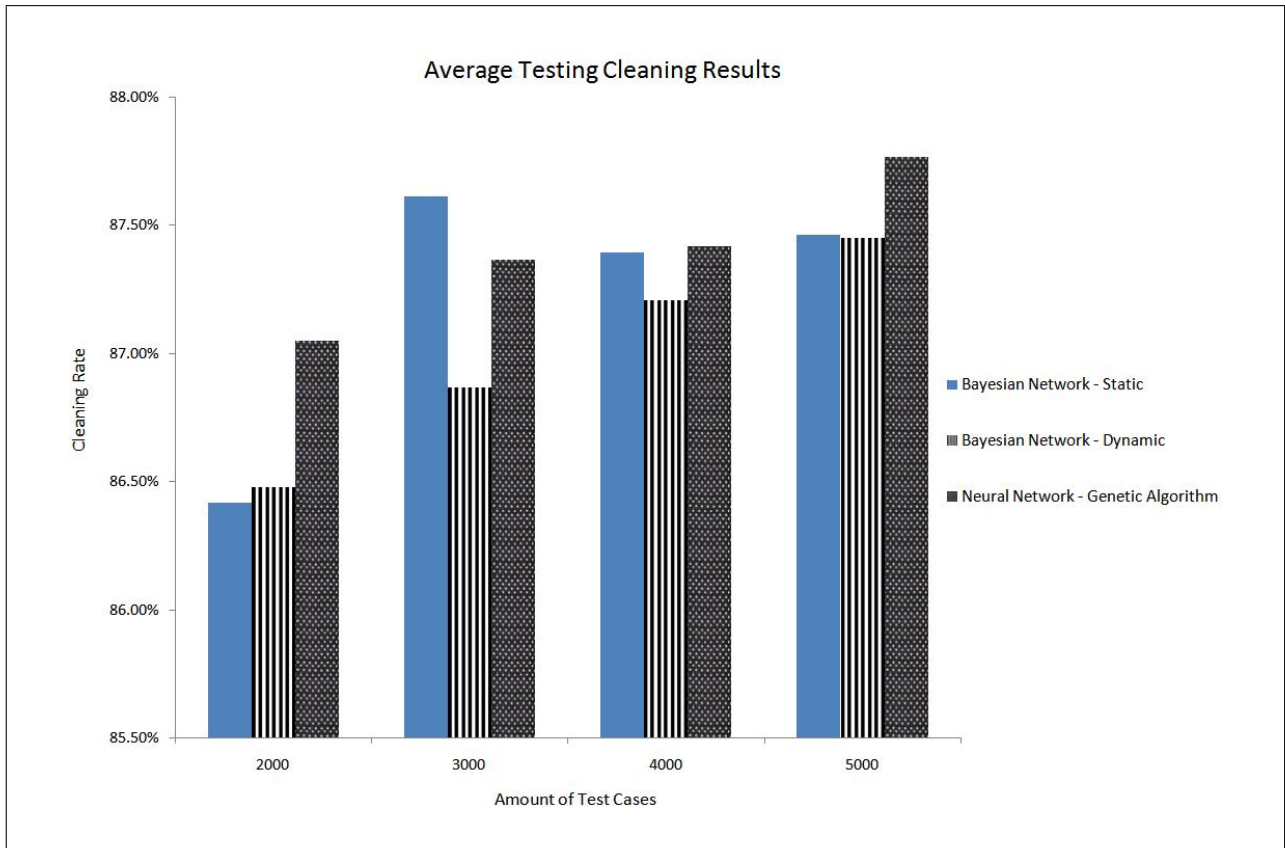


Figure 13: A bar graph depicting the averaged results of the three testing algorithms being utilised to depict the significance of the Neural Network methodology. The solid, striped and dotted bar represents the static bayesian network, dynamic bayesian network and neural network trained by a genetic algorithm respectively.

the first experiment were lower for the 20 chromosome network due to there not being enough generations to find this maximum classification network. The reason we believe why the 100 chromosome was able to find this maximum classifying network in such a short amount of time is that it had the advantage of having five times as much weight configurations to evaluate within its population.

## 6.2 Significance Experiment

The Significance Experiment was created to show that our algorithm can compete with other deferred RFID cleaning algorithms. We utilise the network that achieved the highest record in the Configuration Experiment to compare it to the cleaning rate of two deferred bayesian networks, one dynamic and the other static. Each algorithm has been tested upon four different data sets that contain varying amounts of missing data. These data sets either have 20%, 30%, 40% or 50% missed readings from a 10,000 record data set. The inspiration for the varying amounts of missing data is to properly emulate the ideal that at any given time; passive readers will only record 60%-70% of the total observations. The testing set has been based on the worst possible scenario where only one permutation is correct for each set of data whereas, usually, this occurs minimally.

The configuration that was utilised for this Neural Network was a Genetic Algorithm that was used to train 100 chromosomes for 100 generations. We chose the genetic algorithm due to its high cleaning rate, and chose one that had a relatively more evolved chromosome that the other algorithms. The highest performing algorithm in this experiment was the Neural Network which obtained an 88.36% cleaning rate when correcting 5000 test cases in its first trial.

The algorithm obtaining the least cleaning rate was the static bayesian network which only obtained an 85.95% rate of correctness. As with our previous experiment, we performed three trials upon the algorithms to illustrate how they react to the data.

As with the Configuration Experiment, we have also generated a bar graph to illustrate how effective the average cleaning rates of the three algorithms is. As seen in Figure 13, the highest average cleaning rate was obtained by the Neural Network which was trained by the Genetic Algorithm. Also of note with regard to the highest cleaning is that it occurred when there is 50% missing data which is what we had planned to address due to there being a more likely chance of consecutive missed readings. The lowest achieving cleaning algorithm was both the Static and Dynamic Bayesian Networks. These low readings occurred when there was 20% of the data missing.

Also reflected from the results is that the static bayesian network performed at their peaks during the 30% missing data test cases whereas the dynamic bayesian network performed best cleaning the 50% missing records. Both bayesian networks were beaten in cleaning rate in every test case except for the 30% data set. We believe that this is due to the neural network not being able to clean the data sets with little missing records as effectively as the sets with large amounts of missing data. This is also present within the data set that only contains 20% missing data as the neural network achieved its lowest average performance.

## 7 Conclusion

In this work, we have researched the problem of missed observations within the data warehouse of a

Radio Frequency Identification system utilising intelligent data analysis and an Artificial Neural Network. We have created a methodology that analyses the RFID data to search for missing record anomalies which will have several permutations of possible imputed value sets to be inserted into a data storage. Then, from the utilisation of a neural network that has been trained with a back-propagation algorithm and two genetic algorithms, we determine the most correct permutation to be inserted back into the data storage. We then compared the highest performing network configuration with that of two bayesian networks which have both been trained to correct missing RFID data and found that our neural network obtains a higher cleaning rate.

More specifically, this study makes the following contribution to the field:

- We proposed a deferred methodology that utilises both a Neural Network and intelligent data analysis.
- We applied this methodology to restore missed RFID readings which have not been recorded in the database.
- We studied the accuracy of the resulting networks from two leading ANN training algorithms, back-propagation and genetic algorithms, to determine which would be best suited as an RFID missed readings classifier. This concluded with us finding that the genetic algorithm training provided a superiorly accurate imputed data set.
- We compared our new methodology to two bayesian network methodologies which has also been used to impute missed observations. We found that our ANN algorithm yields a higher cleaning rate especially where the data set is missing approximately half its records.

With regards to future work, we would like to address the application of other training algorithms to our ANN to see if the resulting network will yield a more accurate imputed readings. We would also like to record the effectiveness of the network when the momentum, stopping criteria, hidden units and hidden layers have been manipulated. Additionally, we would also like to attempt to modify the feature extraction process of analysed values used within the Neural Network and possibly apply it to situations outside of RFID applications. Finally, we would like to develop an intelligent data analysis algorithm coupled with the classifiers we have researched to correct other anomalies present within an RFID data set.

## Acknowledgment

This research is partly sponsored by ARC (Australian Research Council) grant no DP0557303.

## References

- Bai, Y., Wang, F. & Liu, P. (2006), Efficiently Filtering RFID Data Streams, in 'CleanDB'.
- Blumenstein, M., Liu, X. Y. & Verma, B. (2007), 'An Investigation of the Modified Direction Feature for Cursive Character Recognition', *Pattern Recognition* **40**(2), 376–388.
- Cha, D., Blumenstein, M., Zhang, H. & Jeng, D.-S. (2008), A Neural-Genetic Technique for Coastal Engineering: Determining Wave-induced Seabed Liquefaction Depth, in 'Engineering Evolutionary Intelligent Systems', pp. 337–351.
- Chawathe, S. S., Krishnamurthy, V., Ramachandran, S. & Sarma, S. E. (2004), Managing RFID Data, in 'VLDB', pp. 1189–1195.
- Collins, J. (2004), 'Boeing Outlines Tagging Timetable [online]', RFID Journal. Available from: <<http://www.rfidjournal.com/article/view/985/1/1>> [Accessed: 5th November 2008].
- Darcy, P., Stantic, B. & Derakhshan, R. (2007), 'Correcting Stored RFID Data with Non-Monotonic Reasoning', *Principles and Applications in Information Systems and Technology (PAIST)* **1**(1), 65–77.
- Darcy, P., Stantic, B. & Sattar, A. (2009a), Augmenting a Deferred Bayesian Network with a Genetic Algorithm to Correct Missed RFID Readings, in 'Malaysian Joint Conference on Artificial Intelligence (MJCAI 2009)', pp. 106–115.
- Darcy, P., Stantic, B. & Sattar, A. (2009b), Improving the Quality of RFID Data by Utilising a Bayesian Network Cleaning Method, in 'Proceedings of the IASTED International Conference Artificial Intelligence and Applications (AIA 2009)', pp. 94–99.
- Floerkemeier, C. & Lampe, M. (2004), Issues with RFID usage in ubiquitous computing applications, in A. Ferscha & F. Mattern, eds, 'Pervasive Computing: Second International Conference, PERVASIVE 2004', number 3001, Springer-Verlag, Linz/Vienna, Austria, pp. 188–193.
- Holland, J. (1975), *Adaption in Natural and Artificial Systems*, University of Michigan Press, Cambridge, MA, USA.
- Jeffery, S. R., Garofalakis, M. N. & Franklin, M. J. (2006), Adaptive Cleaning for RFID Data Streams, in 'VLDB', pp. 163–174.
- Khoussainova, N., Balazinska, M. & Suciu, D. (2007), 'Probabilistic RFID Data Management', *UW CSE Technical Report UW-CSE-07-03-01*.
- McCulloch, W. S. & Pitts, W. (1943), 'A Logical Calculus of the Ideas Immanent in Nervous Activity', *Bulletin of Mathematical Biophysics* **5**, 115–133.
- Rao, J., Doraiswamy, S., Thakkar, H. & Colby, L. S. (2006), A Deferred Cleansing Method for RFID Data Analytics, in 'VLDB', pp. 175–186.
- Raskino, M., Fenn, J. & Lenden, A. (2005), Extracting Value From the Massively Connected World of 2015, Technical Report G00125949, Gartner Research.
- Rooij, A. J. F. V., Johnson, R. P. & Jain, L. C. (1996), *Neural Network Training Using Genetic Algorithms*, World Scientific Publishing Co., Inc., River Edge, NJ, USA.
- Rumelhart, D., Hinton, G. & Williams, R. (1986), 'Learning Representations by Back-Propagating Errors', *Nature (London)* **323**, 533–536.
- Shih, D.-H., Sun, P.-L., Yen, D. C. & Huang, S.-M. (2006), 'Taxonomy and Survey of RFID Anti-Collision Protocols', *Computer Communications* **29**(11), 2150–2166.
- Wang, F. & Liu, P. (2005), Temporal Management of RFID Data, in 'VLDB', pp. 1128–1139.
- Williams, R. W. & Herrup, K. (1988), 'The Control of Neuron Number', *Annual Review of Neuroscience* **11**(1), 423–453.



# Analysis of the Periodical Payment Framework using Restricted Proxy Certificates

Grigori Goldman    Lawrie Brown

School of Information Technology and Electrical Engineering  
University of New South Wales, Australian Defence Force Academy  
Canberra 2600, Australian Capital Territory

grigori.goldman@adfa.edu.au  
lawrie.brown@adfa.edu.au

## Abstract

This paper discusses the design and implementation of a payment framework that is loosely based on the direct debit payment model. We define such payments as one in which customers can authorise merchants to bill them repeatedly for the provision of some service without further interaction with the customers being required. This paper aims to present a first working prototype of our periodical payment model, and to discuss its performance. Our model uses a novel approach for implementing security by employing X.509 restricted proxy certificates over Secure Socket Layer (SSL) to provide authentication, authorisation and non-repudiation services. Although the concept of electronic payments is hardly new and there is a significant amount of interest in improving its security model, most notably from Visa and MasterCard, periodical payments have been consistently overlooked by these industry heavyweights. As of now this concept remains unexplored and the current approaches for securing electronic transactions make it impossible to accommodate this transaction format. The work presented in this paper attempts to fill this niche by developing a new payment specification and a fully working prototype implementation addressing this issue.

**Keywords:** e-commerce, electronic payments, restricted proxy certificates, direct debit, mutual authentication.

## 1 Introduction

Despite considerable research effort by industry leaders like Visa and MasterCard into securing electronic transactions, there still appears to be little published work addressing significant niche topics like the support of periodical payments. In our opinion, most current solutions do not go far enough in addressing issues that are of importance to users of these payment applications. In the context of periodical payments, none of the new frameworks are capable of accommodating this transaction format. This is surprising considering that direct-debit remains one of the most popular payment methods available.

It is clear that current solutions being hastily implemented by industry stakeholders are aimed at fixing the most immediate problems in a quickest and likely cheapest way possible. There does not seem to be any effort spent on making these solutions “future-proof”.

Why is the current approach not sufficient? What limitations does it impose on its use? What alternative applications are needed? These are the questions that will be addressed in this paper in the context of the periodical payment framework.

Before we begin, however, it is important to establish precisely what we mean by this term *periodical payments*. Periodical payments represent direct debit transactions online. They are used for splitting a single logical transaction across multiple physical transfers based on an agreement between the merchant and its customer. This agreement is considered legally binding (although not currently automatically enforceable) on both sides.

Consider the following example. A customer registers with an Internet Service Provider (ISP) to gain access to its Internet broadband service. During the subscription process, the customer provides the ISP with a credit card number, which it can use to charge the customer on a monthly basis. The ISP informs the customer of the amount that will be debited from his account and when this will occur, for example, on the first business day of each month. Once this process is completed, the ISP can debit this account without any further customer intervention.

For registering customers online, this ISP has opted to provide a single payment method option to its customers (i.e. credit cards). While other account types require ISPs to collect signed direct debit request (DDR) forms offline, credit card information can be collected and used without such formalities.

In the present electronic environment this ISP is not alone. Most companies choose credit cards as a payment method to simplify online payments processing. Unfortunately for consumers, this empowers merchants with unprecedented abilities to conduct ad-hoc transactions mimicking direct-debit payments without the backing of formally documented customer consent.

The increased flexibility with relaxed security model offered by credit cards makes the electronic direct debit model vulnerable to abuse. For instance, using the above example, the ISP can easily charge the customer twice within the same month regardless of their initial

agreement that stipulated only one payment transaction per month.

Periodical payments as discussed here are not meant to replace in any way the standard electronic payment mechanisms in place currently. However, the concepts and technologies adopted for its use can be easily generalised to cover all possible electronic transactions.

Furthermore, it might seem unreasonable to expect either Visa or MasterCard to address problems inherent in online direct-debit payments. Existing direct debit applications are not standardised and their use is varied between different merchants and the payment options they choose to provide. This makes solving this issue particularly difficult. However, the argument that we shall make here is that unless provisions for this extra functionality are added now it might not be possible (or at least not simple) to add them in the future. In the next section (Related Work) this topic will be described in more detail with specific focus on the repercussions of the current approaches.

The aim of this paper is to present a prototype of an electronic payment framework that implements a direct debit payment model. This prototype builds on our previous work presented three years ago in Goldman (2007). While that paper presented the theoretical model for periodical payments its weakness was the lack of simulation and test results analysis. This paper aims to fix the shortcomings of its predecessor by strengthening our argument through analysis of initial test data obtained via testing of this prototype. Full details of this work are also provided in Goldman (2009).

A lot has changed in the last three years since the publication of (Goldman 2007). There are more transactions performed over the Internet than ever before. The problems with current periodical payment solutions remain, despite growing interest in electronic payment security. This work hopes to fill this niche.<sup>1</sup>

To minimise the scope, this paper focuses primarily on credit cards, as they are the most widely used and recognised form of online payments. However, it should be noted that the strength of the new approach is in its ability to deliver consistent behaviour across all account types not just credit cards.

The rest of this paper is organised as follows. In Section 2 a brief overview of the current state of academic and industry research in this area is presented. A great deal of attention and focus is dedicated to the discussion of popular approaches taken by the industry in this area and their influence on periodical payments. Section 3 introduces the periodical payment framework specification and discusses its implementation by presenting the prototype. Special emphasis is placed on improvements to the original proposal discussed in (Goldman 2007). In Section 4 a detailed analysis of the performance of the prototype is presented. The focus of this analysis at this stage is on the impact of introducing SSL mutual authentication using restricted proxy

certificates into the framework. Some discussion on using web services as a communication layer is also presented although it is not the main focus here. The paper concludes with a discussion of remaining work (Section 5) and a conclusion (Section 6). Section 7 contains acknowledgements and section 8 references.

## 2 Related Work

With at least three decades of continuous interest in electronic payments, this area has no shortage of related work to use as a starting point. However, given the limited scope of this paper and maintaining its theme, this section will focus primarily on recent proposals that are relevant to credit card payments.

Due to high-complexity of the issue and the extreme difficulty of deploying new concepts and technologies into the market, the pace with which innovation has been introduced into electronic payments has been particularly slow. The frameworks that have been successful at breaching this hurdle are more often than not based on the current technologies and practices (with all of their limitations) rather than being innovative. In fact, the most successful of these systems (e.g. PayPal) are nothing more than proxies into current financial networks.

The industry, however, does at times attempt to break away from the accepted status quo by introducing new and radical changes. Both Visa and MasterCard, for example, are currently in the process of implementing and deploying new and competing authentication technologies aimed at reducing credit card fraud. We will cover these technologies in detail later in this section.

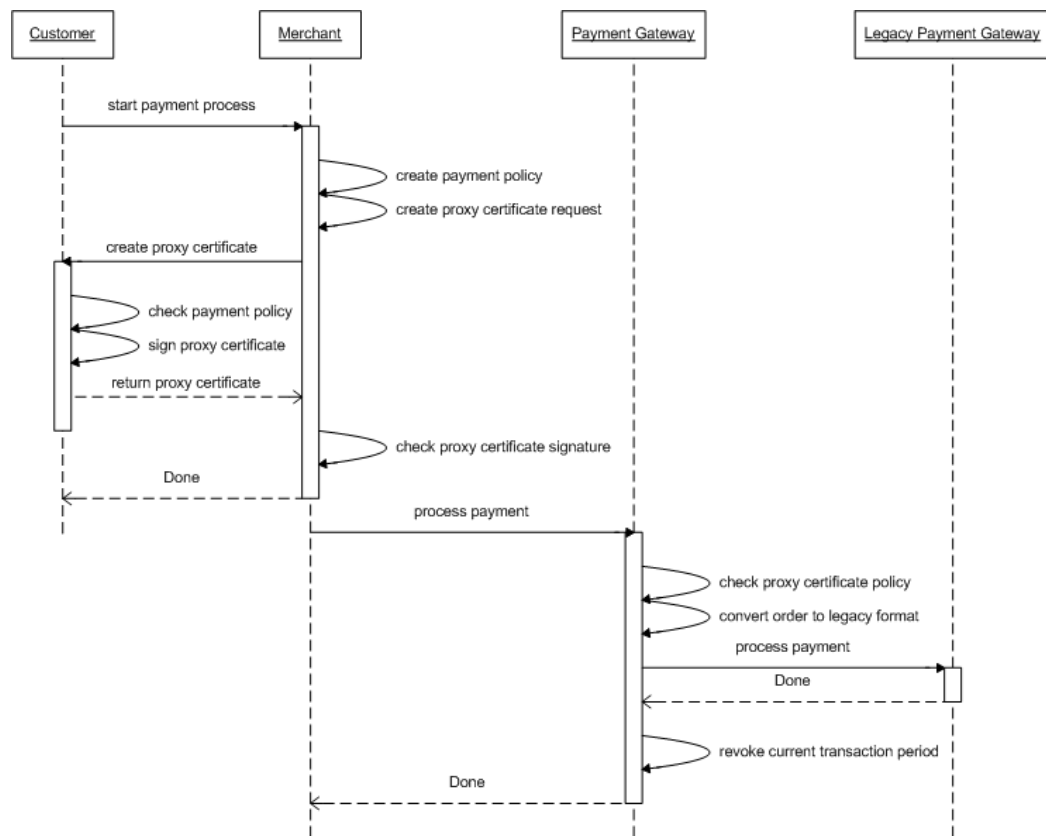
Before introducing competing products solving credit card fraud issues, both Visa and MasterCard (along with several industry leaders) worked together in designing a fundamentally new electronic payment protocol for credit card transactions over the Internet. In the late 1990s a Secure Electronic Transaction (SET) specification was developed. With enormous financial support and initial strong interest in this technology SET was off to a good start.

It has been over a decade since Visa and MasterCard have unveiled SET. However, despite all their effort this technology failed to gain acceptance within the industry. SET was abandoned to the extent that even obtaining references to its original specification documents is becoming difficult. No references to this protocol exist on either Visa or MasterCard web sites and the [www.setco.org](http://www.setco.org) site dedicated to it has been disabled.

There is no doubt that SET was and most likely still is the most ambitious and technically advanced specification dealing with electronic payments. Despite its failure there is a lot that we can learn from it. By studying its design and by analysing the reasons for its failure we can avoid making the same mistakes.

SET specification defines two related phases: 1). registration phase, and 2). purchase phase. Consisting of approximately one thousand pages this protocol is extremely complex and would require an entire paper dedicated to its analysis. As such, only a brief overview will be presented here (see MasterCard, Visa 1997a and 1997b for details).

<sup>1</sup> Legal issues surrounding the use of digital certificates and legislative requirements for electronic payments are complex and best left to the experts. As such, these aspects are considered out of scope of this paper.



**Figure 1: Proxy Certificate Delegation and Payment Process**

The registration phase of the SET protocol deals with how both customers and merchants obtain cryptographic credentials that they can use for authentication and authorisation purposes. This is a mandatory step that both parties have to complete before they can participate in SET transactions. Registering for SET involves using a dedicated Certification Authority (CA) capable of issuing digital certificates on behalf of the card brands.

The purchase phase of the SET protocol is a more interesting although unnecessarily complicated process. Just like the framework presented in this paper, SET is concerned with payments only, rather than the entire shopping experience. As such, it assumes that by the time it is invoked the customer selections have already been propagated to the merchant.

The crux of this phase involves the use of client-side software. It initiates the payment process, creates order (OI) and payment instructions (PI) and performs various cryptographic and communication services. An important part of this process is protecting the OI from the payment gateway and the PI from the merchant using a dual signature concept introduced in (MasterCard, Visa 1997a). A dual signature is constructed by separately hashing OI and PI constructs and then concatenating the hashes, hashing them again and signing them with the customer's private key.

When the merchant receives the request it contains the OI and the hash of PI as well as the customer's signature. The PI itself is encrypted using a key only available to the payment gateway. The merchant can verify the signature without having access to the contents of PI. Conversely, by giving the payment gateway the encrypted PI and the

hash of OI it can also verify the signature without gaining access to the OI.

The most interesting aspect of the SET protocol directly relating to periodical payments is its concept of instalment payments. Instalment payments allow a merchant to split a single large payment over multiple smaller transactions. It works by adding an optional construct into the PI, which contains the terms for each instalment similar to the proxy policies discussed later on. Since normally PIs cannot be used multiple times the payment gateway issues a separate authentication token (e.g. reuse token) to the merchant for the next scheduled transaction. Once the last payment is processed no more authentication tokens are issued.

There is no doubt that SET was ahead of its time. Despite all the effort dedicated to its development it unfortunately ended in failure. There are numerous reasons why it failed. For example, both the registration and the purchase phases of the protocol were extremely complicated requiring over a thousand pages of formal specification to properly describe them. This was frequently criticised in (Giampaolo et. al. 2002 and 2003, Paulson 2002, Stallings 2006).

In addition, SET required the use of downloadable client-side software, which contributed to frequent shopping cart abandonment. This, of course, is less relevant now due to the increases in bandwidth available to Internet users.

Also, its overuse of digital envelopes at every stage of the process, requiring generation of multiple symmetric keys, followed by continuous verification of signatures made SET unacceptably slow. For example, Wolrath (1998) described how some vendors reported lag times of

up to fifty seconds for processing a single cardholder request. The solution presented later on avoids these performance issues by reducing the complexity of the protocol and minimising the use of cryptography.

As was already mentioned both Visa and MasterCard have abandoned SET. Instead they have introduced alternative, competing technologies aimed at solving credit card fraud issues. For Visa this technology is Visa Three Domain (3-D) Secure authentication protocol, while MasterCard implemented a Secure Payment Application (SPA) protocol. A good discussion of each protocol is provided by GPayments (2002).

Visa 3-D Secure uses browser redirection as the main mechanism by which cardholders are authenticated. Merchants redirect users to their issuers, which perform authentication and redirect the user back to the merchant site. This works particularly well using username and password but can also be adapted for use with smartcards.

Unfortunately, credential caching is impossible with browser redirection. As such, the cardholder must be authenticated for every transaction (meaning the customer needs to re-enter credentials over and over again). MasterCard SPA solves this issue using a client side applet.

Unlike SET, a SPA applet does not require any customer specific information hence it can be safely installed and used on even unsecured machines (e.g. Internet cafes, etc). Cardholders do need to enter issuer information before making purchases as it is needed during authentication.

Using an applet instead of browser redirects means that customer credentials can be cached locally and reused over multiple payment sessions. Once shopping is completed the credentials are destroyed and another user can safely reuse the SPA applet.

The design of Visa 3-D Secure and MasterCard SPA has a significant impact on the way periodical payments can be implemented. Despite their ability to use strong, two-factor authentication via digital certificates, both protocols are essentially session based. They provide no support for delegation and in fact require the customer to be physically present during every transaction.

Using the current approach taken by both vendors it is not possible to implement periodical payments since this particular transaction type precludes customer participation after the initial credential exchange.

### 3 Periodical Payment Framework

#### 3.1 X.509 Proxy Certificate Definition

X.509 certificates are used to solve authentication and authorisation problems. A certificate binds a public key to a user identity, while the private key is kept secret. Using the private key, the user may prove to other parties that he is the owner of the certificate.

It is clear that reliance on the private key as proof of ownership of the certificate can be problematic if the user wishes to delegate some of his privileges to other parties. Without revealing the private key, this is not possible using the conventional X.509 certificate.

The proxy certificate extensions attempt to overcome this problem by providing mechanisms for delegating certificates to other parties without compromising the

user's private key. These extensions are described in (Tuecke et. al., 2004).

Restricted proxy certificates allow issuers to define restrictions, which are placed on proxy certificates via policies. This enables issuers to define which subsets of their overall permissions are to be transferred to another party, thus subsequently reducing the potential damage that can be caused by credential misuse.

#### 3.2 Periodical Payment Model Specification

Unlike SET that described a complete cardholder and merchant registration process, our periodical payment framework deals solely with payments. It assumes that cardholders and merchants will be issued with digital certificates separately (e.g. when being issued credit cards, etc). This makes our specification a lot simpler to describe and analyse.

In this section we shall present a brief overview of the theoretical model that underpins periodical payments. For a more comprehensive description refer to our previous work in Goldman (2007).

Periodical payment model consists of two separate processes. The first is the delegation of permissions by the customer to the merchant using restricted proxy certificates. The second is the actual payment transaction, initiated by the merchant that uses a proxy certificate to authenticate itself to the payment gateway on customer's behalf and perform the funds transfer.

The delegation of the restricted proxy certificate is the only step in this process that involves the customer. It is based on a well-known X.509 proxy certificate delegation protocol defined in (Tuecke et. al., 2004). It works by requiring the merchant to generate a proxy certificate request containing the policy, which is then forwarded to the customer. Once the customer signs this request, it becomes the proxy certificate used for initiating payment transactions.

The periodical payment policy used within the proxy certificate is the core of our payment model. It is used to encode the contractual agreement between the customer and the merchant and is subsequently used to verify legitimacy of each payment transaction. It is encoded as an XML document within the proxy certificate and generally declares one or more assertions that restrict the use of the certificate and the amount that a merchant can charge the customer.

Since proxy certificates are self-contained and are not co-signed by either the merchant or the payment gateway, we have used an attribute within the policy, *merchant-dn*, to bind the certificate to a specific merchant. This protects the certificate from being stolen and used by an unauthorised party.

The syntax of the policy language is simple. The <pay> element encapsulates most of the contractual information discussed in this paper. This assertion encodes both the contract boundaries (i.e. begin and end dates) as well as the interval between each transaction.

The example XML in Table 1 defines a simple policy that allows a merchant to charge a customer account a fixed amount of ten Australian dollars. The "on" attribute specifies when each transaction can occur. The syntax for this attribute is based on a common UNIX time-based scheduling service called CRON. In this particular

example the CRON expression states that the merchant may initiate a transaction on the first workday of every month in 2009.

```
<payment-policy merchant-dn="cn=ISP,..."
  payment-gateway-dn="cn=PaymentGateway,...">
  <source-account>
    <creditcard cardnumber="xxx"
      cardholder="name" expiry="2011-10"/>
  </source-account>
  <pay currency="aud" amount="10"
    on="0 0 * 1W * ? 2009"/>
</payment-policy>
```

**Table 1: Periodical Payment Policy XML**

A customer will receive this policy as part of a proxy certificate request generated by the merchant. It is customer's responsibility to check the terms of the agreement and if acceptable sign the request thus creating a proxy certificate which is then returned to the merchant. Once received, the merchant can verify the proxy certificate signature to ensure that it matches a valid cardholder. Provided that the signature is valid the proxy certificate can then be used to initiate payment transactions according to the payment policy. This process is depicted on the left hand side of Figure 1.

The process of initiating a payment transaction is simple. Using mutual authentication over Secure Socket Layer (SSL) protocol with X.509 restricted proxy certificate as the authentication credential, the merchant can unambiguously identify itself to the payment gateway and at the same time prove that it has rights to access a customer's account. The advantage of using proxy certificates is that they are virtually identical to the standard X.509 certificates currently used for SSL. This means that this protocol can be used unmodified with restricted proxy certificates as authentication credentials.

Once authentication is completed and a secure channel is established the payment gateway can retrieve the payment policy from the certificate and use its assertions to determine the validity of the payment instruction it received from the merchant. The process of verification is also simple. The payment gateway must ensure that the amount being transferred is within the accepted limit. In addition, it will check that the time of the transaction matches the CRON expression inside the "on" attribute of the <pay> element. Provided that the transaction is valid, the payment gateway performs the transfer using the existing banking network and sends an acknowledgement message back to the merchant containing a transaction receipt. This process is likewise depicted in Figure 1. The key point to note is that merchant to payment gateway communication occurs without any customer intervention.

### 3.2.1 Double Charging Problem

There is a significant flaw in the above process. It assumes that given a policy, the payment gateway will be able to unambiguously determine whether a transaction is valid. In actual fact, it cannot. By only comparing a policy to the current payment instruction received from the merchant, it is not possible to detect double charging. That is, the payment gateway can only verify whether

payment instructions match a policy but cannot say with any degree of certainty that the order is unique and has not been already processed previously.

This is one of the most fundamental issues that prompted our initial interest in this topic. The success of this framework is dependent on finding an appropriate solution to this problem.

The solution that we have adopted is based on the concept of certificate revocation as described in (Housley et al., 2002). Normally, certificate revocation lists are used to revoke compromised certificates. However, in our case proxy certificates cannot be revoked, as merchants will be unable to initiate any more payment transactions using them. Instead, the payment gateway can use the payment period defined within the policy to revoke the use of a certificate for the immediate payment period. For example, given a monthly payment period (as in Table 1), after processing a transaction the payment gateway would make an entry in the transaction revocation list (TRL) revoking the use of the presented certificate until the next month. Any attempt to use the same certificate twice within the same month would fail due to this entry in the list. The following is an example of such an entry in the TRL revoking the use of a certificate in the month of March. This entry is derived from the policy in Table 1.

```
revoke="* * * 1W MAR ? 2009"
```

### 3.2.2 Cancelling Periodical Payments

One important advantage of being able to electronically create and sign periodical payment policies is the ease with which such agreements can be cancelled when things go wrong. There are numerous reasons why policies may need to be cancelled. Due to long-term nature of these types of payments it is only natural to assume that customers may want to terminate agreements when their terms are no longer suitable (e.g. merchant competitors are offering better deals for the same type of services, etc).

Currently, cancelling direct debits once they are established is difficult. The control over this process rests in the hands of the merchants. Usually, it is not in their best interest to allow customers to terminate contracts before they expire due to the loss of potential income. As such, a lot of merchants make it difficult for customers to cancel agreements.

The periodical payment framework that we developed provides a solution to this problem by implementing a cancellation process based on an XML cancellation policy embedded within the overarching payment policy.

The merchant is responsible for encoding the conditions under which policies can be cancelled by making a special entry (i.e. <cancellation-policy>) within the payment policy XML. For example, Table 2 depicts an extension to our original payment policy listed in Table 1. In this example, two separate <pay> assertions are used to declare two distinct fee structures, which are dependent on when the policy is cancelled. Terminating the policy within the first six months in 2009 carries with it a higher cancellation fee of twenty dollars, while in the last six months this fee is halved.

Using this policy a customer may cancel the periodical payment agreement by communicating directly with the

payment gateway using its URL (as encoded in the cancellation policy). The cancellation request must contain the original, signed proxy certificate and must be authenticated using the same certificate that was used to create the proxy. Using this information the payment gateway will be able to establish whether the cancellation request is legitimate. Provided that cancellation is permissible within the current payment period as per the policy, the payment gateway will process this request.

```
<payment-policy>
...
<cancellation-policy url=http://pg/cancel-service>
  <pay currency="aud" amount="20"
    on="* * * * 1-6 ? 2009"/>
  <pay currency="aud" amount="10"
    on="* * * * 7-12 ? 2009"/>
</cancellation-policy>
</payment-policy>
```

**Table 2: Periodical Payment Cancellation Policy XML**

In our scheme the policies are not immediately cancelled when customer requests are processed. Since requests are most likely to be received in the middle of a payment period, merchants are entitled to a final transaction, such as receiving any outstanding amounts and charging fees. As such, the payment gateway simply registers each request (i.e. policy) in a cancellation registry to be processed next time that certificate is used by the merchant, at which time the merchant is given an opportunity to finalise the account.

This scheme makes a number of assumptions about how cancellation of periodical payments will be commonly used. These assumptions compromise between flexibility, functionality and simplicity of the protocol. For example, this scheme does not support immediate termination of contracts. This means that given a monthly payment period, a customer cannot suspend payments halfway through the month. In this case, this approach favours the merchant as it can collect more revenue before the contract is permanently cancelled. On the other hand, the merchant using this scheme is not aware that a policy has been cancelled until an attempt to debit a customer is made. Furthermore, the merchant must react immediately when it receives a pending cancellation message, hence all its administrative tasks must be easily automated. This may not appeal to some merchants, for example, merchants whose customer retention strategies yield good results.

Finally, to improve security this scheme binds the policy to a payment gateway (see *payment-gateway-dn* attribute in Table 1). This ensures that the merchant cannot switch payment gateways to bypass a registered cancellation. Unfortunately, this eliminates the possibility of load balancing or failover between different payment gateways. We recognise this as a limitation of the scheme and plan to address this in the future.

### 3.3 Payment Framework Implementation

The architecture of the periodical payment framework consists of three distinct tiers: 1). client-side browser extension, 2). merchant payment processing service, and 3). acquirer payment gateway web service.

Using a browser extension to implement the client-side software needed for enabling proxy certificate delegation is a logical choice. Browser plug-ins and extensions are becoming increasingly common way of distributing custom software to Internet users. This trend is partially motivated by the growing popularity and development of the Mozilla Firefox browser. This browser is built on top of a mature, extensible framework that makes it simple to dynamically extend its graphical user interface and behaviour.

Utilising a combination of XML User Interface Language (XUL), JavaScript and Java we have developed a simple prototype extension that captures HTTP requests carrying base64 encoded proxy certificate requests and returns signed proxy certificates. For now this extension looks for a user's keystore on the file system when signing proxy certificates, however, in practice Firefox provides features for accessing user credentials directly from smartcards.

The business logic is implemented using a combination of JavaScript and Java. JavaScript is used for processing HTTP requests and sending HTTP responses while all cryptographic operations are performed in Java. An optimised version of this extension would see Java being replaced with a more efficient XPCOM/C++ implementation.

The merchant software is implemented as a Java 2 Enterprise Edition (J2EE) web application running on Apache Tomcat servlet container. The communication between the merchant application and the client-side browser extension is handled by a servlet filter which encodes the logic for generating proxy certificate requests and processing proxy certificates returned by client browsers. This filter is designed to cache the new proxy certificates and their corresponding public/private key pairs for use by the merchant application (i.e. when it is ready to use them for initiating payment transactions).

Normally, a single Java web application would only use one digital certificate for SSL authentication. As such, a Java application can be configured at runtime with a keystore location using the following system property: `javax.net.ssl.keyStore`. This keystore is accessed during SSL mutual authentication process.

For periodical payments this default Java behaviour is insufficient, as a single Java process (i.e. merchant server) must dynamically change digital certificates when processing payments for different customers. To overcome this limitation a custom security provider and key manager was implemented that is capable of dynamically selecting digital certificates and their keys according to the currently executing customer payment. It can do this even in a multi-threaded environment. These credentials are obtained from a cache managed and updated by the servlet filter described previously.

Specifically, the merchant payment process is forced to use restricted proxy certificates when exchanging SOAP messages with the payment gateway web service. This web service was configured to be accessible only via HTTPS protocol (i.e. over SSL secured channel). Furthermore, mutual authentication was also enabled requiring the merchant to present a valid X.509 certificate during the SSL handshake.



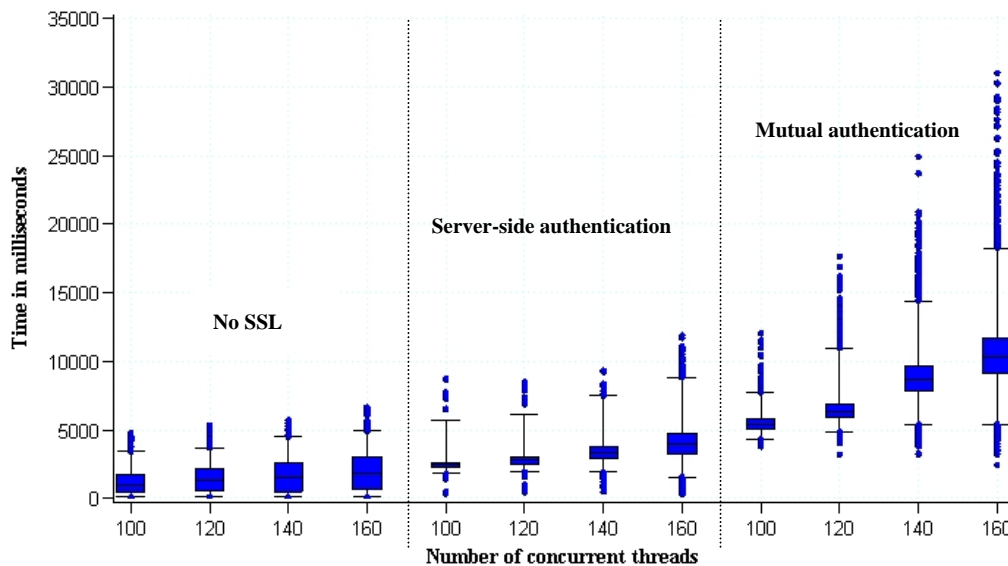


Figure 2: Total request processing time

Just like the merchant, the payment gateway application is deployed on Apache Tomcat servlet container. It uses Apache Axis2 as a web services engine that implements the entire web services feature stack.

During integration testing we discovered that the standard Java SSL socket implementation does not handle proxy certificates. It rejects proxy certificates because they contain an end-entity certificate within the certificate chain. In other words, it assumes that only CAs can issue and sign certificates. Due to time constraints we used a “work around” by adding a basic CA constraint to customer certificates thus in a fact making each customer a CA.

This is not a significant issue since there are alternative configurations that can make Tomcat recognise proxy certificates. Instead of using a Java Secure Socket Extension (JSSE) as an SSL provider, for example, we can use an Apache Portable Runtime (APR) that directly integrates with OpenSSL (SSL implementation that supports proxy certificates). Considering that OpenSSL is significantly faster than JSSE, changing SSL implementations should deliver better performance results making this change our top priority as we move forward.

#### 4 Performance Analysis

A quantitative approach was chosen for testing the periodical payment prototype. Having little reference data to start with our aim was to collect enough empirical data to make reasonable conclusions regarding its overall performance. It is reasonable to assume that other electronic payment applications can benefit from the data collected here since many new applications are beginning to follow the same approach. For example, PayPal and Google Checkout are both introducing web services over SSL.

In this section we shall focus on the performance of the merchant to the payment gateway communication. We consider this part of the framework to be more critical than the relatively low volume interactions between the client browser extension and the merchant. This exchange

is of course equally important; however, due to its infrequent use even relatively poor performance is likely to be acceptable.

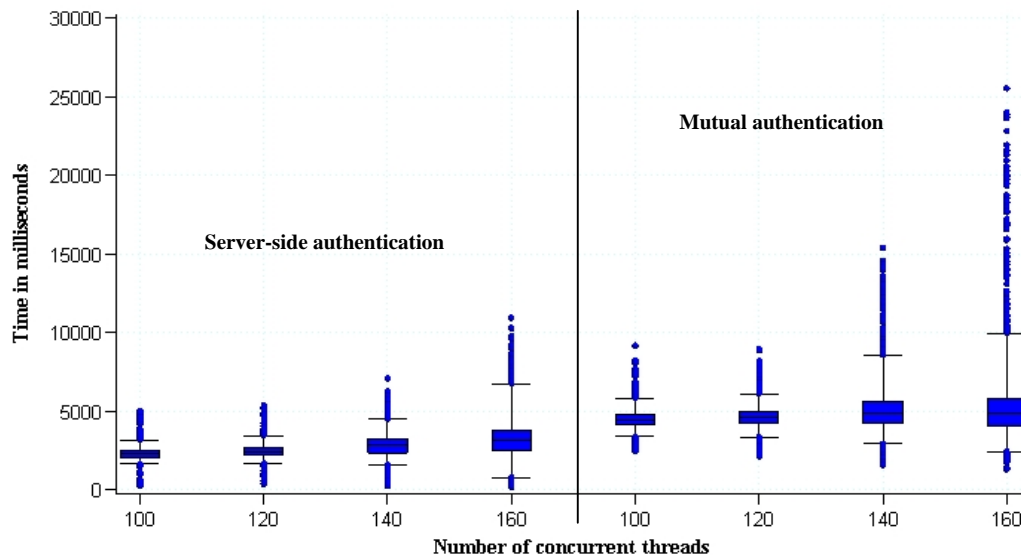
Overall, four independent tests were executed with an increasing number of concurrent threads accessing the payment gateway web service. Specifically, the merchant service created 100, 120, 140 and 160 concurrent threads each one generating 100 independent payment requests thus producing in total 10000, 12000, 14000 and 16000 unique web service requests.

Each test was initialised with a set of pre-generated unique proxy certificates needed for SSL mutual authentication. Unique certificates were needed to ensure that the merchant or the payment gateway could not cache credentials and reuse them for multiple sessions. Each certificate contains a unique distinguished name and a policy whose amount value is also different to other policies within the test set.

Unfortunately, it was not possible to broaden the spectrum of tests by increasing the number of concurrent threads (or the number of requests per thread) due to the limitations imposed by the hardware. Any attempts to increase the load on the payment gateway caused crashes of the JVM. It remains an exercise for the future to re-run these tests using more capable hardware, that better reflects that likely to be used by merchants and payment gateways.

To be able to accurately measure the true impact of adding mutual authentication with restricted proxy certificates to the framework the same tests were executed with no SSL and with SSL server-side certificate authentication only.

The test configuration with no SSL authentication was useful for establishing a baseline/benchmark to compare other results with. On the other hand, server-side certificate authentication test is needed since this form of authentication is prevalent on the Internet today. Comparing the results of these tests against our proposed solution should make it easy to determine whether using certificate delegation for client-side authentication is a



**Figure 3: SSL handshake processing time**

viable approach for solving the periodical payment problem.

#### 4.1 Test Results

It is reasonable to assume that adding SSL to the framework will have a significant impact on its performance regardless of whether it uses server-side only or mutual authentication. Furthermore, we predict that increasing the load on the server by raising the number of concurrent threads will have a similarly detrimental effect. At best, we are expecting linear performance degradation as the number of simultaneous connections grows.

The results of executing the three test scenarios described in the previous section (i.e. no SSL, server-side and mutual authentication) are summarised in Figure 2 using a box plot. We have chosen to use a box plot, as it is a convenient and efficient way of summarising a large data set. It provides a visual representation of the most important aspects of a distribution (e.g. median value, inter-quartile range measuring statistical dispersion and lower/upper outliers).

Specifically, Figure 2 plots total request processing time of executing each request. An individual data point represents request processing time starting from when the merchant first connects to the payment gateway up to and including receipt of a response message upon completion of a transaction.

The results presented in Figure 2 are consistent with our prediction that the growth in request processing times is linear. This observation can be mathematically substantiated using Pearson's Product Moment Correlation Coefficient (PMCC). A PMCC value of 0.9975 (for no SSL authentication), 0.9877 (for server certificate authentication), and 0.9895 (for mutual authentication) reveals an almost perfect linear correlation between the number of simultaneous connections and the performance of the server. The fact that this is happening consistently regardless of the changes to SSL configuration increases our confidence that our observations are accurate.

Unfortunately, increasing the number of simultaneous requests impacts data dispersion. The inter-quartile range (IQR) within each data series rises with the number of concurrent threads. Containing 50% of all requests, these IQR measurements indicate an increase in volatility as the server is overloaded. It should be noted, however, that this in itself is not unusual. As the load on the payment gateway increases, environmental factors are starting to play a more significant role on performance of individual requests thus creating variability in test results (e.g. garbage collection, page swapping, etc).

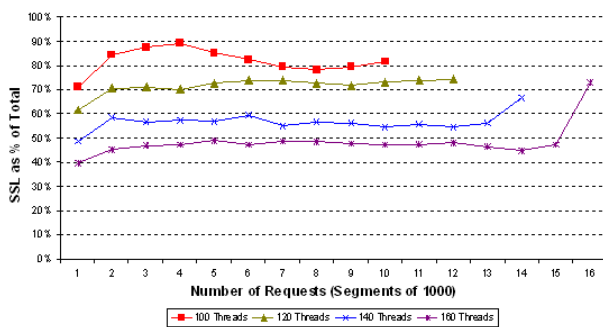
Figure 2 also reveals that there is a substantial increase in request processing time using mutual authentication. This difference is much higher than the original variation between no SSL and server-side certificate authentication only scenarios.

In Figure 3 the impact of mutual authentication on performance is measured in relation to the standard server-side certificate authentication. Unlike Figure 2, which depicts the total processing time for each individual request, this graph shows only the SSL handshake part of the overall request execution logic. By isolating the SSL handshake process we eliminate the ambiguity introduced by other parts of the application such as the web services engine that could have skewed the results.

Figure 3 follows the pattern established in Figure 2. The data represents a distribution with a flat curve and a positive skew. This data is also similarly dispersed with the number of concurrent threads having the same impact as before. That is, increasing the spread of data over a larger time set. However, the rate with which performance degrades does not match. Figure 2 displays a much higher rate of decay than Figure 3 (87 milliseconds per additional thread versus 10 milliseconds). This imbalance suggests that other factors unrelated to mutual authentication are affecting performance.

We can confirm that SSL is not solely responsible for degrading performance as Figure 2 suggests by examining the ratio of SSL handshake to total time it takes to process a single request. In Figure 4 this ratio is described as the percentage of the difference between

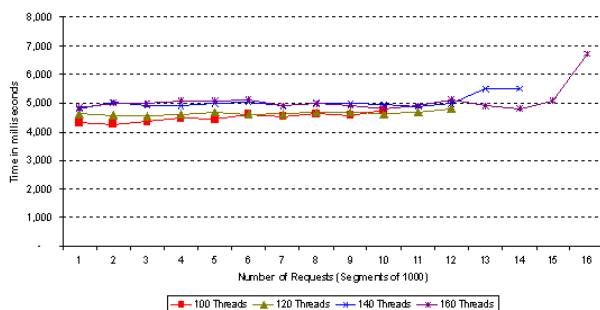
SSL handshake and the total time. This data is plotted using the average percentage over segments of 1000 requests.



**Figure 4: SSL handshake percentage of total time**

Predictably, Figure 4 clearly shows that as the number of concurrent threads grows the percentage of the total time taken by the SSL handshake actually drops. This would not be the case if SSL was contributing to this extra processing time. In that case we would also see an increase in the percentage as well. This behaviour proves that adding mutual authentication to the periodical payment protocol does not have any unexpected side effects.

In fact, it is surprisingly easy to prove that the SSL handshake remains stable and consistent under increased server load. Plotting the averages using segments of 1000 requests on a time series chart (see Figure 5) reveals an unanticipated lack of variance between the different data series. Some points on this graph interlace showing how similar the times are despite increases in concurrent threads.



**Figure 5: Average SSL handshake processing time**

Having proven that SSL is not solely responsible for the sharp increase in processing times observed in Figure 2, we must find an alternative source of this variance. The only other significant component of the periodical payment framework that can potentially impact performance is the Apache Axis2 web services engine. This engine provides all of the “boilerplate” code required to implement a complete, standards compliant web services application. It performs all SOAP message handling logic and handles the entire web services communication. It is likely that this part of the application is causing the observed behaviour. However, it remains an exercise for the future to isolate exactly which component of this engine is causing performance issues.

The test results reveal that the impact of SSL with mutual authentication can be considered modest and reasonable. The benefits of using mutual authentication

clearly outweigh the relatively small increase in performance figures. Considering that the behaviour of the application under load is stable we are not overly concerned with the initial results. Since SSL authentication can be implemented in hardware (see next section for further discussion) it is clear that this part of the application is unlikely to cause issues.

The other significant component of the application, the web services engine, is clearly underperforming. There is a significant amount of time spent within the web services communication layers. What is more disappointing is that as the load on the server increases the performance sharply degrades which is quite opposite to the SSL figures that displayed remarkable consistency. Fortunately, alternative implementations or technologies can be adopted to replace the Apache Axis2 web services engine if required.

## 5 Future Work

Detailed analysis of the periodical payment framework highlighted some issues with performance of both the SSL handshake and the web services communication components. While results look promising, a proper scalability test is required to ensure that this concept will scale to realistic load volumes. Before this can happen we need to isolate individual components of the web services stack to determine which part of the architecture is causing performance issues.

There are two related issues that need to be addressed regarding web services. The first is whether the web services concept is appropriate in the electronic commerce context. Web services provide interoperability and decoupling but do their extensive performance overhead justify their use? Our goal is to investigate alternative technologies that would provide a similar level of interoperability without the extra overhead.

As a second goal, we aim to evaluate alternative web services implementations. Focusing primarily on performance, the aim is to compare the Apache Axis2 engine to other frameworks offering the same features. It is quite likely that by switching to another web services provider performance gains can be achieved without any code changes.

Some performance improvements are also needed for the SSL handshake. There are various options at our disposal to make this part of the framework much faster. At present, Apache Tomcat servlet container running the payment gateway application is configured using JSSE. JSSE is a pure Java implementation of SSL, and as such it suffers performance issues. By integrating an alternative library developed using native APIs we can dramatically decrease the times taken by the SSL handshake. One such approach is to configure OpenSSL open source SSL library into Apache Tomcat using its Apache Portable Runtime (APR) mechanism.

Alternatively, an SSL hardware accelerator could be used to test the payment gateway in a production-like setting. A hardware accelerator is a special-purpose device designed to perform the SSL handshake in hardware thus delivering the best possible performance. These devices are commonly used by organisations with high Internet traffic that rely on SSL for authentication. While the cost of this device might prove prohibitive, it is

none the less part of our agenda to investigate its use within the periodical payment framework.

In addition, the issue of customer credential storage should also be addressed. At the moment, our extension is loading a customer's keystore from a file. This is sufficient for testing but would hardly be acceptable in a final product.

In the past credential storage was an insurmountable issue. While smartcard technology has been around for many years, even most modern computers now are not equipped with smartcard readers needed to make this technology available for Internet use. This, however, is no longer a problem. Alternative technologies exist right now that make smartcard use a reality.

One such technology is USB keys. A USB key combines both the smartcard and a smartcard reader in a single device. The advantage of using USB keys is that USB is extremely popular with even relatively old computers supporting this standard. It is our goal for the future to integrate USB token support into our Firefox extension thus providing complete end-to-end implementation of periodical payments.

Another potentially promising technology that requires further research is using Subscriber Identity Module (SIM) cards commonly found in mobile phones as cryptographic tokens. SIM cards are essentially smartcards that are capable of storing customer cryptographic credentials. By integrating mobile phones into the periodical payment model we can leverage of the existing technology and infrastructure providing an alternative avenue for electronic commerce.

## 6 Conclusion

Periodical payment framework delivers a solution to an existing problem affecting users of the direct debit payment model. It solves issues of past implementations by leveraging of the currently available, standards compliant and industry supported technologies. At the same time it does not compromise on either security or performance.

It is clear that existing approaches to electronic commerce by industry leaders, such as Visa and MasterCard are not directly addressing this particular payment format. In fact, current attempts rely on customers' physical presence during payment transactions making delegation impossible. With the growing popularity of direct debit payments there is a clear need for a more secure way of conducting such payments over the Internet.

In this paper we have demonstrated that a periodical payment framework built using web services and secured using SSL with restricted proxy certificates is a viable solution to this problem. While using SSL with mutual authentication does impact performance, we have proven that this increase is insignificant when measured against the benefits it provides.

The use of web services has delivered mixed results. While we have benefited from its flexibility there are still questions regarding its performance. However, the data analysed so far is promising. Even under load, the payment gateway is performing admirably with linear performance degradation as the number of simultaneous requests grows.

There are numerous options still available to improve performance of both the SSL and web services components. As such, there is reason for optimism in the future of this approach.

## 7 Acknowledgements

We would like to thank Milutin Pribicevic for his help with analysing the test results. His insightful comments and suggestions were instrumental in preparing the performance analysis section of this paper.

## 8 References

- Giampaolo, B., Lawrence, P., Massacci, F. (2002): The Verification of an Industrial Payment Protocol: The SET Purchase Phase. *Proc. 9<sup>th</sup> ACM Conference on Computer and Communications Security (CCCS2002)*, Washington, DC, USA, 12-20, ACM Press.
- Giampaolo, B., Lawrence, P. and Massacci, F. (2003): Verifying the SET Registration Protocol. *IEEE Journal of Selected Areas in Communications*, **21**(1):77-87
- Goldman, G. (2007): Periodical payment model using restricted proxy certificates. *Proc. Thirtieth Australasian Computer Science Conference (ACSC2007)*, Ballarat, Australia, **62**:131-139, ASC Inc.
- Goldman, G. (2009): Periodical payment framework using restricted proxy certificates. *PhD Thesis*. Currently submitted for examination.
- GPayments (2002): Visa 3-D Secure vs. MasterCard SPA: A comparison of online authentication standards, GPayments Pty Ltd.  
[http://www.gpayments.com/pdfs/GPayments\\_3-D\\_vs\\_SPA\\_Whitepaper.pdf](http://www.gpayments.com/pdfs/GPayments_3-D_vs_SPA_Whitepaper.pdf). Accessed 27 July 2008.
- Housley, R., Polk, W., Ford, W., Solo, D. (2002): Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. <http://www.ietf.org/rfc/rfc3280.txt>. Accessed 14 August 2009.
- MasterCard, Visa (1997a): SET Secure Electronic Transaction Specification: Business Description. <http://hostedfiles.110mb.com/2007/11/set-secure-electronic-transaction.html>. Accessed 27 July 2008.
- MasterCard, Visa (1997b): SET Secure Electronic Transaction Specification: Programmer's Guide. <http://citeseer.ist.psu.edu/289529.html>. Accessed 27 July 2008.
- Paulson, L. (2003): Verifying the SET Protocol: Overview. *Formal Aspects of Security*. **2629**:4-14, Springer-Verlag, Berlin.
- Stallings, W. (2006): Web Security: Secure Electronic Transaction. In *Cryptography and Network Security: Principles and Practice*. 4<sup>th</sup> ed., Prentice Hall.
- Tuecke, S., Welch, V., Engert, D., Pearlman, L., Thompson, L. (2004): RFC3820: Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile. <http://www.ietf.org/rfc/rfc3820.txt>. Accessed 4 August 2008.
- Wolrath, C. (1998): SET: A market survey and a test implementation of SET technology. Accessed 27 July 2008. <http://www.wolrath.com/set.html>



# Automated Functionality Testing through GUIs

Duc Hoai Nguyen, Paul Strooper, Jörn Guy Süß

School of Information Technology and Electrical Engineering  
The University of Queensland  
Queensland 4072, Australia

{ducnh,pstroop,jgsuess@itee.uq.edu.au}

## Abstract

*Model-based GUI testing* (MGT) is emerging as a promising approach for testing applications with a *graphical user interface* (GUI). Currently, test models in MGT approaches are close to the GUI implementation with limited ability to represent abstract actions. This paper introduces the Action-Event Framework (AEF), a MGT framework. This framework helps testers abstract away from low-level details of the GUI under test and generate test cases in a behaviour-oriented way. In this framework, testers can perform both business logic testing and GUI testing in a reusable manner. At the core of AEF is a mapping language that allows test engineers to map abstract actions to GUI implementations. The paper proposes several coverage criteria based on links between abstract actions and event sequences. Tool support is provided for several steps of the framework. To evaluate AEF, a case study on a task manager is conducted to determine the time necessary to test the GUI, the types of defects that can be detected, and the correlation between the proposed coverage criteria and code coverage.

**Keywords:** GUI testing, model-based testing.

## 1 Introduction

Today, many software products provide GUIs to end users in the form of a web-based or window/dialog interface. However, despite the widespread use of GUIs, GUI testing in practice is still fairly ad hoc (Memon 2002). In this paper we use *GUI testing* as a shorthand for functionality testing by using the GUI of the *system under test* (SUT) as the interface.

In manual GUI testing, testers analyse requirements, design test cases and execute them (Perry 1995, Hetzel 1988). System responses are observed and compared with expected outputs to determine test verdicts. A first step to automate this procedure is the use of test scripts (Fewster and Graham 1999). Test scripts are programs that automate test steps. They are typically written in scripting languages or in the implementation language of the SUT. Test scripts can also be produced automatically by *capture and replay tools* (CRTs) such as CompuWare TestPartner, IBM Rational Robot, Mercury WinRunner, and Segue's SilkTest (Li and Wu 2004, Hartman 2002).

These tools record interactions between the tester and the GUI, and support the capturing of screens for later comparison. They generate test scripts that record steps. The recorded information is usually positional (e.g. click on button A at the screen coordinate X,Y) and thus fragile to GUI changes. During test execution, they replay the previously recorded GUI events by executing the scripts and judge success by the appearance of an expected captured screen. CRTs have significant maintenance issues, in that whenever the GUI layout changes, steps affected by the changes may need to be re-captured and re-integrated with the existing test by editing scripts (Finsterwalder 2001, Li and Wu 2004, Daboczi et al. 2003). In general, CRTs only reduce some of the effort of completely manual test script development and do not result in significant savings (Li and Wu 2004).

A number of research results have shown *model-based testing* (MBT) as a promising solution to overcome the maintenance weakness of CRTs (Neto et al. 2007, Utting and Legeard 2007). In MBT, the tester typically builds a formal model which captures behaviour of the SUT and generates test cases from that model (Utting and Legeard 2007).

Some research proposals have attempted to apply MBT to test GUIs (Paiva 2007, Alsmadi and Kenneth 2007, Kervinen et al. 2006, Andrews et al. 2005, Memon et al. 2003b, Memon 2001, Reza et al. 2007, White and Almezen 2000). In this paper, they will be referred to as *model-based GUI testing* (MGT) approaches. These approaches suggest testing GUIs by using models that represent events and event interactions. However, due to the complexity of the models in these approaches, the modelling effort is considerable. Moreover, these models are dedicated to GUI testing while ignoring potentially available models and test cases for the underlying business logic.

We introduce AEF, a MGT framework which enables test engineers to model both abstract actions and GUI events. Abstract actions are modeled in an action model and mapped to GUI events via a mapping model. The GUI events are recorded in an event collection called the *GUI model*, which provides detailed information about the events. The mapping model, in contrast, focuses more on the structural information and the order between events. To build the mapping model, AEF offers a mapping language to define how actions are implemented in the GUI. AEF aims to save testing effort in three ways:

- Test GUIs in a more manageable way: AEF allows testers to develop test models and generate test cases in a behaviour-oriented manner. Section 3 presents coverage criteria (Utting and Legeard 2007) which

specify how much of the action and the mapping model is covered by the generated test cases.

- Reuse BL test models and test cases: logical defects can originate from either the business-logic in the underlying application or the GUI programming in the event handlers. AEF can be used for both GUI testing and BL Testing. During BL Testing, the action model is used to generate BL test cases and uncover business-logic defects in the underlying application. BL test cases are later reused in *GUI Testing* to generate GUI event-level test cases to uncover logical defects in GUI programming.
- In AEF, because the business logic is decoupled from GUI events, any GUI changes will affect only the GUI model and the mapping model, while the action model is still up-to-date. This helps reduce the cost of maintaining the test models.

The contributions of this paper include the testing framework, an action-to-event mapping language, novel coverage criteria, a preliminary effectiveness evaluation of the framework on a small but real system, and prototype tool support.

The rest of the paper is organised as follows: Section 2 describes related work on MGT. Sections 3 introduces the proposed Action-Event Framework and compares it with existing approaches. A case study is presented in Section 4. Section 5 draws conclusions and addresses future work.

## 2 Related Work

This section reviews MGT approaches and discusses how these approaches model GUI behaviour. Memon et al. (Memon 2001, Memon et al. 2003b) propose an event-based modelling method. The GUI is decomposed into components. Events within each component are represented by an *event flow graph* (EFG). A node in an EFG is a GUI event. A transition indicates that an event can occur after another. The inter-component interactions are modeled by an *integration tree* (IT) (Memon et al. 2003b). EFGs and ITs are built automatically with a reverse-engineering tool that generates the test models from the GUI implementation (Memon et al. 2003a). The problem of modelling GUI data is not addressed. To generate test cases, testers have to specify initial and goal states of the GUI. Test cases are auto-generated by chaining pre/post-conditions of events between the initial and the goal states. This means that testers have to define the pre/post conditions for all events. This burden can be relieved in regression testing, in which an original GUI is used as a test oracle. To determine the test oracle of a test case, the test case is executed on the original GUI, and the resulting GUI state is used as the test oracle.

Kervinen et al. (2006) propose the manual modelling of abstract actions in an action machine. Each action is refined by a refinement machine which defines how an action can be performed at the GUI event-level. Both action and refinement machines are represented as *labeled transition systems* (LTS). To generate test cases, the actions in the action machines are replaced by corresponding refinement machines to obtain a composite LTS. Test cases are generated from this composite LTS.

Andrews et al. (2005) divide web-based GUIs into subsystems, each modeled by a FSM. Each FSM consists of nodes representing webpages or form objects, with transitions representing navigations. Navigation between subsystems is captured in a system-level FSM. Another type of test model is a decision tree (Strelzoff and Petzold 2003). A decision tree can be reverse engineered from the GUI source code using static semantic analysis.

Behaviour of the SUT depends not only on what events are being invoked by the user, but also on the event data. For example, a textbox can typically accept arbitrary strings. The value of the string can affect the behaviour of the GUI, complicating the test models. Manual development of such data-driven models is painful. None of the approaches described above address this problem. One solution for this problem can be found in recent MBT approaches which employ a set of action functions (Paiva 2007, Campbell et al. 2005). Action data is associated with actions via function parameters. The action state machine is automatically generated through an exploration algorithm which triggers actions with given parameter values and observes how the state of the system changes correspondingly. In this way, testers do not have to manually model the action state machine, especially the states resulting from different action input values. Testers only need to define actions and parameter value sets. In this paper, this approach is called *parameterized action modelling* (PAM).

Spec Explorer (Campbell et al. 2005) is a typical PAM tool. This tool employs a modelling language called Spec#. It has been applied to MGT by Paiva et al. (2007). To reduce the modelling effort, a graphical front-end is developed which allows testers to describe GUI behaviour in UML diagrams. These diagrams are later transformed into a Spec# program which consists of empty action functions (Campbell et al. 2005). Each action function represents a GUI event. Testers have to complete the function bodies to define the semantics of the events.

## 3 The Action-Event Framework

The previous section has explained how PAM aims to overcome the GUI data modelling problem. However, we believe that PAM-based approaches can be further improved by reducing modelling effort. Even a moderately sized GUI like WordPad has up to 121 events that can be triggered (Memon 2001), leading to substantial modelling effort to model GUI behaviour. Modelling effort can be saved if we avoid defining event semantics for every event.

This paper proposes the *Action-Event framework* (AEF), another PAM-based approach. It is a two-layer approach. At the top layer is an action model which defines abstract actions. At the bottom layer is a mapping model, which maps abstract actions to sequences of concrete GUI events that implement the actions. An example of an abstract action in MS WordPad is opening a file which can be implemented as a sequence of GUI events such as click on menu File, click on menu item Open, and so on. As there are far fewer abstract actions than GUI events, the effort for defining an action model is also less than for defining an event model. However, in



AEF, extra costs are incurred for developing the action mappings. However, the evaluation in Section 4 shows that the overall cost can be less than the traditional PAM-based approach.

Figure 1 depicts the AEF workflow. The components of this architecture are described below.

**Requirement specification:** a description of intended system behaviour, normally written in natural language.

**GUI under test:** the GUI being tested.

**Action model:** MBT requires a formal model of application behaviour. This model is commonly built by translating the textual requirements specification into a formal model. The model is typically a form of state machine in which states represent anticipated states of the SUT and transitions represent actions that move the system from one state to another. From such a state machine, action-level test cases are generated. So far, AEF has used Spec# (Campbell et al. 2005), a pre/post modelling language. An example Spec# action model is presented in Figure 2. The details of this action model are explained later in this section.

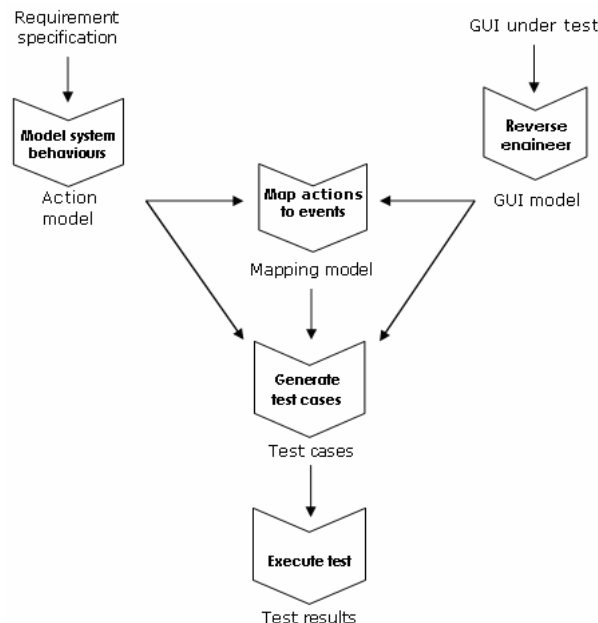


Figure 1. Action-Event Framework

**GUI model:** a GUI model can be automatically reverse engineered from the GUI using dynamic or static analysis techniques. It is a list of widgets with associated events and attributes. In static analysis, it is generated from the GUI source code and does not capture any dynamic interactions between widgets. This can be a problem on some types of GUIs, for example when widgets are generated dynamically. Dynamic analysis techniques overcome this problem by recording information about the GUI at runtime.

**Mapping model:** the mapping model links actions in the action model to events in the GUI model. In other words, the mapping model defines how abstract actions are implemented in the GUI. This step is similar to the procedure of building test adaptors in traditional MBT. The difference is that a traditional test adaptor connects a

model with source code, whereas in AEF a mapping model connects an action model with a GUI model, which represents the structure of the GUI. The mapping model is described programmatically using extensions to the Spec# language, which are discussed in detail in Section 3.1.

**Test cases:** for applications with GUIs, the number of possible scenarios or test cases is usually infinite. In AEF, the generation of test cases is guided by coverage criteria. Structure-based coverage criteria such as state coverage and transition coverage can apply to either the action model or the mapping model. As an abstract action can be linked to many event sequences in the mapping model, AEF also introduces coverage criteria that specify how the mapping model is covered.

**Test results:** the generated test cases can be executed online or offline to produce test results (Utting and Legeard 2007). With online testing, test case generation and execution are performed in an interleaving manner. The generation process can hence respond to volatile parameters returned by previous steps. In contrast, with offline testing, test cases are executed only after the completion of test case generation. This improves performance but requires a higher degree of predictability of the underlying SUT.

Compared to existing approaches (Paiva 2007, Campbell et al. 2005, Alsmadi and Kenneth 2007, Andrews et al. 2005, Kervinen et al. 2006, Memon 2001, Memon et al. 2003b), AEF has the following potential advantages:

- By replacing detailed event modelling with action and mapping modelling, we believe the overall modelling effort will be reduced.
- Actions can be mapped to various permutations of events.
- Test cases are generated in a behaviour-oriented way.
- While the business logic is defined in the action model, the implementation details are part of the mapping model. Therefore any changes in the GUI implementation affect only the mapping model.
- The action model can be used to test the underlying business logic, then re-used to generate GUI-level test cases.

The last advantage in the list leads to testing effort savings. Usually, during the development process, the underlying business logic is developed before the GUI front-end. In AEF, the action model is developed before the mapping model, hence can be used for testing the business logic. When development of the GUI front-end is completed, testers only need to develop the mapping model and convert the BL test cases into event-level ones. The reuse of the action model and BL test cases in GUI testing results in effort savings and helps early detection of defects in the underlying business logic. This is in contrast to existing GUI testing approaches, in which the test models are dedicated for GUI testing.

### 3.1 The mapping model

In this section, we present how a mapping model is specified using AEFMAP, a mapping language which maps actions to GUI events.

A BNF definition of the language is given in Figure 3. Below we explain the symbols of this language.

**Mapping model:** a mapping model consists of a number of mapping functions.

**Mapping function:** a map from an abstract action to event sequences that implement the action.

**Function parameters:** a list of parameters of a

mapping function. The signatures of the mapping function must match the signature of the corresponding action. Hence, both must have the same name and parameters. This suggests that all data types, including built-in types and user-defined types, that appear in the action signatures must be supported by the mapping language.

```
// type declaration
enum Progress {New, Finished, Working}
class Task {
    string name;
    Progress progress;
}

// declare a ToDo list as a sequence of tasks
type ToDoList = Seq<Task>;
// declare a ToDo list variable and initialize it
ToDoList todolist = Seq{};

// create a new task. By default, the task name is empty.
[Action]int newtask()
{
    Task t=new Task("", Progress.New);
    todolist=todolist.Add(t);
    return todolist.Size;
}

// edit the ith task
[Action]Task edittask(int i, string name, Progress progress)
{
    todolist[i].name=name;
    todolist[i].progress=progress;
    return todolist[i];
}

// delete the ith task
[Action]int deletetask(int i)
{
    todolist=todolist.Take(i)+todolist.Drop(i+1);
    return todolist.Size;
}
```

Figure 2 An example action model

<mapping-model>	::=	<mapping-function> <mapping-function> < mapping-model >
<mapping-function>	::=	<function-signature> "{" <function-body> "}"
<function-signature>	::=	<return-type> <function-name> "(" ( <param-list>   "" ) ")"
<param-list>	::=	<param-type> <param-name>   <param-type> <param-name> "," <param-list>
<function-body>	::=	<event-map> <return-statement>
<event-map>	::=	<event-execution>   <seq-generator> "{" <event-executions> "}"
<seq-generator>	::=	"Serialize"   "Select"   "Permute"
<event-executions>	::=	<event-execution>   <event-execution> <event-map>   <event-map> <event-execution>
<event-execution>	::=	<exe-keyword> "(" <event-name> "," <event-input> ")" ","
<exe-keyword>	::=	"Execute"   "ExecuteOp"

Figure 3 Definition of the mapping language

```

// create a new task either by clicking on the menu item or the toolbar
int newtask() {
    Select {
        Execute(GUI.menuTODOADD.click);
        Execute(GUI.toolbarADD.click);
    }
    return GUI.tree.Size;
}

//edit a task by selecting the task, updating task information, then clicking on the Allow button.
Task edittask(int i, string name, Progress progress){
    Serialize{
        Execute (GUI.tree.select(i));
        Permute{
            ExecuteOp(GUI.textboxNAME.type, name);
            ExecuteOp(GUI.comboboxPROGRESS.select, progress);
        }
        Execute(GUI.buttonALLOW.click);
    }
    return new Task( GUI.tree.Node(i).Text,
                    GUI.comboboxPROGRESS.SelectedItem);
}

// delete a task by selecting the task, then clicking the menu item or the toolbar.
int deletetask(int i) {
    Serialize{
        ExecuteOp(GUI.tree.select(i));
        Select{
            Execute(GUI.menuTODODELETE.click);
            Execute(GUI.toolbarDELETE.click);
        }
    }
    return GUI.tree.Size;
}

```

Figure 4 An example mapping model

**Function body:** the body of a mapping function includes an event map and a return statement. A return statement is a statement which calculates the return value of the mapping function based on observed GUI attributes.

**Event map:** an event map specifies how event sequences can be formed from a group of events. It can be nested so that event maps can occur inside other event maps.

**Sequence generator:** events in a group can form event sequences in three different ways, depending of which sequence operator is used. The current sequence operators are *Serialize*, *Permute*, and *Select*.

**Event execution:** the execution of a single event.

Figure 4 shows example mapping functions. This example is taken from the case study presented in Section 4.

#### Using the mapping language to map actions to event sequences

The basic elements of AEFMAP are the *Execute* and *ExecuteOp* functions. They invoke single GUI events.

*ExecuteOp* is used for optional events that can be skipped, while *Execute* is used for mandatory events.

An action is typically mapped to sequences of events, not a single event. Given an abstract action with input values and expected outputs, testers need to explicitly specify the GUI events and the order in which the events are triggered to achieve the expected outputs. Note that the order of events can affect the expected output. Therefore, AEFMAP introduces three operators that operate on groups of events: *Serialize*, *Permute*, and *Select*.

**Serialize:** requires sequential execution of events.

**Permute:** requires execution of all events in any order.

**Select:** requires execution of exactly one event from a group.

The operators can be nested, providing flexibility to express various combinations of events. Figure 4 presents an example mapping model for the actions defined in Figure 2. It indicates that the action *edittask* is mapped to the following five GUI event sequences:

- tree.select → textboxNAME.type → comboboxPROGRESS.select → buttonALLOW.click

- tree.select → comboboxPROGRESS.select → textboxNAME.type → buttonALLOW.click
- tree.select → textboxNAME.type → buttonALLOW.click
- tree.select → comboboxPROGRESS.select → buttonALLOW.click
- tree.select → buttonALLOW.click

### Mapping action input and output data to GUI attributes

Testers generate action-level test cases by supplying action data in the action model. Action data can be manually derived from system requirements or generated automatically (Ganov et al. 2007). Therefore, an action-level test case includes not only an action sequence, but also inputs and expected outputs of each action in the sequence. In AEF, this input and output data is mapped to concrete GUI attributes via glue code written in AEFMAP.

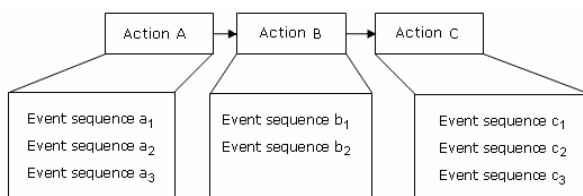
An action's input data is mapped to GUI attributes by writing AEFMAP code to transform input data of the action to appropriate values and supplying these values to *Execute* and *ExecuteOp* function calls. In the example in Figure 4, the second parameter of *Execute* and *ExecuteOp* is optional and allows testers to specify input data for the events. For example, *comboboxPROGRESS.select* has the input *progress*; the value 0 for *progress* means the task is "Active" and the value 1 means "Finished".

Output data of actions at GUI attribute-level is expressed as return expressions in mapping functions. These return expressions define how the actual output of the actions is calculated from the runtime values of GUI attributes. In Figure 4, the attributes of the *Task* object returned from the action *edittask* are mapped to the label of the current node in the tree and the value of the combo box *Progress* of the GUI.

While AEFMAP is simple, it is powerful enough to express relations between abstract actions and GUI events, as shown in the case study in Section 4. It is declarative and abstracts away procedural programming issues and uses a minimal set of operators that should be easy to understand and learn.

## 3.2 Test coverage criteria

Test coverage criteria control the number of test cases generated. As previously stated, they address coverage in terms of the action and the mapping models.



**Figure 5. The generation of event sequences**

Figure 5 illustrates the relation between an action-level test case generated from an action model and GUI-level test cases derived from that action-level test case. Leaving out the input data and expected outputs, an

action-level test case is a sequence of actions. Similarly, without test inputs and expected outputs, a GUI-level test case is a sequence of events. So, when discussing the mapping model coverage criteria below, we use the terms action sequences and event sequences instead of action-level test cases and event-level test cases.

An action model represents a state machine. Action sequences are generated from this action state machine. These sequences are transformed into event sequences based on the action-event mapping defined in the mapping model. By concatenating event sequences of individual actions, AEF can form different event sequences that implement an action sequence.

### Action-based criteria:

Action model coverage is addressed using traditional coverage criteria such as state coverage or transition coverage. These coverage criteria are widely used in MBT. These criteria are reused in AEF. They specify how much of the action state machine is covered. Mapping model coverage is more specific to AEF, so will be described in more detail below.

### **Definitions:**

#### Action-event links:

For a mapping function  $f$  that maps an action  $ai$  to a set of event sequences  $f(ai)$ , the event sequences in  $f(ai)$  are said to be the action-event links of the action  $ai$ .

#### Link-based criteria:

These coverage criteria focus on the action-event links between the action model and the GUI model. These are the criteria specific to AEF and include Action-One-Link, Action-All-Link, and Action-N-Way.

**Action-One-Link coverage:** this criterion requires that, for each action, only one action-event link is covered. This means the number of generated event sequences is equal to the number of action sequences. This criterion is quite weak in terms of code coverage and event interaction coverage, since it does not necessarily cover all action-event links. However, it is useful in smoke testing. Smoke testing is normally the first test performed after integration or modification to provide some level of assurance that the system under test works with some typical actions. Therefore, in smoke testing, only some typical tests are executed.

**Action-All-Link coverage:** This criterion requires that, for each action, all action-event links are tested.

**Action-N-Way coverage:** The Action-All-Link coverage criterion can be considered as a one-way coverage criterion over the sets of action-event links because it covers all links of individual actions. So, it can be generalized to the *Action-N-Way* (ANW) coverage criterion, which requires the coverage of all possible combinations of action-event links of  $N$  actions. Depending on the value of  $N$ , this criterion has many variants such as 2-way (pairwise) coverage, 3-way coverage, 4-way coverage, etc. A special case of ANW is when  $N$  is equal to the number of actions in the action-level test case. This results in the Cartesian product of the set of action-event links associated with the actions. This type of coverage is called Cartesian Coverage.

In the definitions above, we have introduced coverage criteria for the action model and the mapping model. A complete coverage criterion should consider both models, so should be a combination of an action-based criterion and a link-based criterion. For example, testers can define a coverage criterion which combines the All-Transition Coverage criterion for the action model with the Action-All-Link coverage criterion for the mapping model.

### 3.3 GTG – The Framework Prototype

We have developed the *GUI Test Generator* (GTG), a prototype tool that generates test cases from a given action model, GUI model, and mapping model (Figure 6). GTG requires the action model and the GUI model to be provided in the form of XML files, while the mapping model is a plain-text file.

The GUI model is generated from the GUI using *Quick Test Pro* (QTP), which records widget types and attributes while a user navigates between widgets of the GUI under test (dynamic reverse engineering).

The mapping model is written in the AEFMAP language. We plan to build an *Action-Event Recorder* (AER) tool to support creating mapping models. When mapping a particular action, testers perform the action on the GUI under test and use AER to record all user interactions in the form of AEFMAP code. We believe that AER can help reduce mapping effort, especially when testing large or complex GUIs.

Test cases are generated in the form of QTP test scripts, which can be executed automatically in QTP.

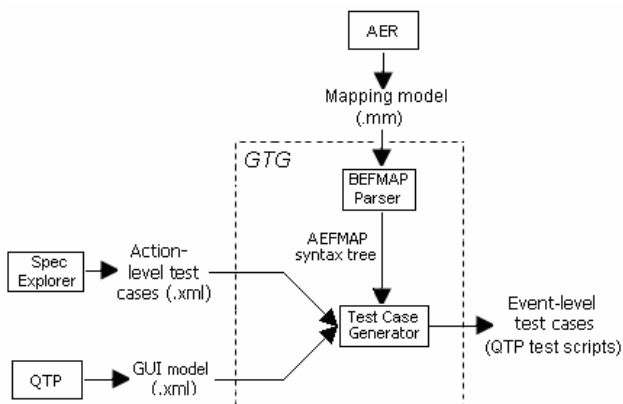


Figure 6 GTG architecture

## 4 Case Study

We illustrate AEF by applying it to To Do Manager (Thommen 2008), an open-source GUI application that allows users to manage a list of tasks, as can be seen in Figure 7. The GUI has been modified (e.g. some features are dropped, a toolbar is added) so that we can illustrate various aspects of AEF. Six actions of To Do Manager have been tested: *create new tasks*, *edit tasks*, *delete tasks*, *create new task lists*, *save task lists*, and *open existing task lists*.

### 4.1 Mapping between actions and events

The six tested actions of To Do Manager are defined in a Spec# action model. From this model, Spec Explorer generates 35 action-level test cases. Table 1 shows one of these test cases. This action-level test case will be used

later in this section to illustrate the generation of event-level test cases.

The action model for the To Do Manager is shown in Figure 2. For the sake of brevity, it shows only three actions: *newtask*, *edittask*, and *deletetask*. In this model, a task is modeled by a user-defined type called *Task*, which consists of a *task name* and *task progress*. *Task progress* can take one of three values: *New*, *Finished*, and *Working*. The task list of To Do Manager is monitored through a variable of a type called *ToDoList*. This type represents a sequence of *Task* objects. From this action model, Spec Explorer generates an action state machine. Action-level test cases are generated from this state machine using coverage criteria such as all-state-coverage or all-transition-coverage. The latter is used in this case study.

The action-level test cases are used to test the underlying logic of To Do Manager. They are also later reused in GUI testing. To generate event-level test cases, actions need to be mapped to events via a mapping model. The mapping functions for *newtask*, *edittask*, and *deletetask* are presented in Figure 4. These mapping functions refer to GUI events and attributes of the To Do Manager GUI model, which was reverse engineered from the GUI by QTP.

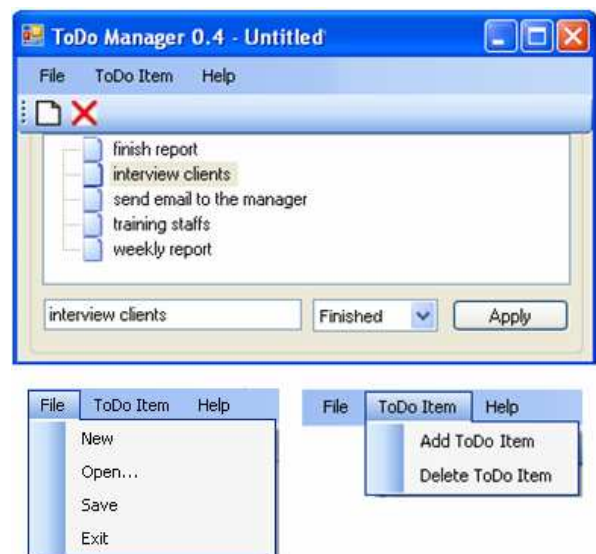


Figure 7 To Do Manager user interface

Table 1 Action-level test case 6

Action-test case	Actions	Expected outputs
ATC6	newtask()	1
	edittask(0,"abcd", Progress.Finished)	Task("abcd", Progress.Finished)
	deletetask(0)	0

#### Mapping actions to event sequences

*Newtask* is mapped to two events, as users can perform this action by clicking either the menu item *Add To Do Item* or the toolbar button *Add*. *Deletetask* is



performed by selecting a task, then clicking on the menu item *Delete To Do Item* or clicking the toolbar button *Delete*. *Edittask* is more complicated. The mapping function *edittask* indicates that users can edit a task by selecting it, typing a task name into the *TaskName* text box, selecting a progress value from the *Progress* combo-box, and clicking the *Allow* button. However, the second and the third events of this sequence can be interchanged or omitted.

To Do Manager is a straightforward GUI application, so its mapping model is simple. In this case study, the mapping functions contain only one- or two-level nested structures of sequence operators. Larger applications will contain more complicated nested structures.

#### Mapping action inputs and outputs to GUI attributes

In the mapping model of To Do Manager, GUI inputs are calculated from action inputs and supplied to the events through parameters of the *Execute* or *ExecuteOp* function calls. For the action *edittask*, the action inputs *name* and *progress* are supplied to the two events *textboxNAME.type* and *comboboxPROGRESS.select* respectively. Testers can also write AEFMAP code to perform calculations on these inputs before supplying values for event parameters.

For action output mapping, the return expressions of *newtask*, *edittask*, and *deletetask* define how the actual outputs of these actions are calculated from the run-time values of GUI attributes. For example, attributes of the *Task* object returned from the action *edittask* are mapped to the label of the current node in the tree and the value of the combo-box *Progress*.

## 4.2 Link-based coverage criteria and the generation of event-level test cases

GTG performs three steps to convert the action-level test case ATC6 to event-level test cases:

- Mapping the action sequence *newtask*  $\rightarrow$  *edittask*  $\rightarrow$  *deletetask* to event sequences.
- Mapping input data of actions to input data of events.
- Mapping expected outputs of actions to expected outputs of events.

From the mapping model of To Do Manager, GTG builds a list of event sequences associated with each action of the action sequence *newtask*  $\rightarrow$  *edittask*  $\rightarrow$  *deletetask*, as shown in Table 2.

In the event sequences *Edittask1* to *Edittask5*, it can be seen that the two events *textboxNAME.type* and *comboboxPROGRESS.select* are present in some sequences but absent in others, because they are optional events. Moreover, their position in these sequences can be swapped, due to the *Permute* operator in the mapping function *edittask*.

Below, we present how various coverage criteria affect the generation of event-level test cases.

**Action-One-Link coverage:** This coverage criterion requires one action-event link to be tested for each action. If there are multiple action-event links associated with an action, then any sequence can be selected. When generating Action-One-Link test cases, GTG can be configured to choose the first sequence for each action or

to choose one sequence randomly. For example, if testers configure GTG so that the first event sequences are chosen, then it generates one event sequence as follows: *Newtask1*  $\rightarrow$  *Edittask1*  $\rightarrow$  *Deletetask1*.

**Table 2 Action-Event links of actions in ATC6**

Action: Newtask	
Newtask1	menuTODOADD.click
Newtask2	toolbarADD.click
Action: Edittask	
Edittask1	tree.select $\rightarrow$ textboxNAME.type $\rightarrow$ comboboxPROGRESS.select $\rightarrow$ buttonALLOW.click
Edittask2	tree.select $\rightarrow$ comboboxPROGRESS.select $\rightarrow$ textboxNAME.type $\rightarrow$ buttonALLOW.click
Edittask3	tree.select $\rightarrow$ textboxNAME.type $\rightarrow$ buttonALLOW.click
Edittask4	tree.select $\rightarrow$ comboboxPROGRESS.select $\rightarrow$ buttonALLOW.click
Edittask5	tree.select $\rightarrow$ buttonALLOW.click
Action: Deletetask	
Deletetask1	tree.select $\rightarrow$ menuTODODELETE.click
Deletetask2	tree.select $\rightarrow$ toolbarDELETE.click

**Table 3 Test case generation using A2W coverage**

TC1	Newtask1 $\rightarrow$ Edittask1 $\rightarrow$ Deletetask1
TC2	Newtask1 $\rightarrow$ Edittask2 $\rightarrow$ Deletetask2
TC3	Newtask2 $\rightarrow$ Edittask3 $\rightarrow$ Deletetask1
TC4	Newtask2 $\rightarrow$ Edittask4 $\rightarrow$ Deletetask2
TC5	Newtask1 $\rightarrow$ Edittask5 $\rightarrow$ Deletetask1
TC6	Newtask2 $\rightarrow$ Edittask1 $\rightarrow$ Deletetask2
TC7	Newtask2 $\rightarrow$ Edittask2 $\rightarrow$ Deletetask1
TC8	Newtask1 $\rightarrow$ Edittask3 $\rightarrow$ Deletetask2
TC9	Newtask1 $\rightarrow$ Edittask4 $\rightarrow$ Deletetask1
TC10	Newtask2 $\rightarrow$ Edittask5 $\rightarrow$ Deletetask2

**Action-All-Link coverage:** it is obvious that there is more than one way to combine eight event sequences of *newtask*, *newtask*, and *edittask* so that all the action-event links are covered. The minimum number of generated event-level test cases is equal to the largest number of links between actions. In the case of BTC6, there would be at least 5 event-level test cases required to cover all action-event links, because *edittask* has the most links (5).

**Action-N-Way coverage:** For pair-wise coverage (ANW with N=2), the 10 combinations for the action sequence *newtask*  $\rightarrow$  *edittask*  $\rightarrow$  *deletetask* shown in Table 3 are sufficient. B3W requires the Cartesian product of the three actions, hence requires  $2 \times 5 \times 2 = 20$  test cases.

## 4.3 AEF and the traditional PAM approach

This section compares code coverage, defect-detection ability, and testing effort between AEF and the traditional PAM approach. By “traditional PAM”, we mean using Spec Explorer to model GUI events without the presence of a graphical front-end, like the one proposed by Paiva et al. (Paiva 2007). Obviously, AEF can employ a similar front-end to generate a skeleton of the action model, but that is out of the scope of this paper.

Using PAM, we built a Spec# program that models GUI events of To Do Manager so that this program



covers the same actions and events as the AEF's action and mapping models presented in Section 4.1 (6 actions, which cover 32 GUI events). Table 4 shows that the PAM approach results in a large Spec# model while AEF produces a much smaller action model. Note that the source code size of To Do Manager is 1805 lines of code (1805 LOC). Even though AEF requires extra effort for defining the mapping, that effort is significantly less than the effort for defining an event-based model. The problem is not just the large number of events to be modelled, but also the difficulty in linking the event semantics to the requirements and the cost of debugging the model.

**Table 4 Modelling effort To Do Manager**

	Size of test models (LOC)	Testing effort (hrs)
AEF	Action model: 92 Mapping model: 98	Build action model: 5.5 Build mapping model: 2.5
PAM	Event model: 519	Build event model: 23

**Table 5 Code coverage and defect-detection comparison**

	Coverage criteria	Code coverage	Defects detected
AEF	One-Link Coverage	51.4%	9
	All-Link (1-way) Coverage	77.3%	13
	Cartesian coverage	78.2%	14
PAM	All-transition	78.2%	14

To measure the defect-detection ability of AEF, we asked volunteers to inject 17 artificial defects in the implementation and checked which defects were detected by the two approaches. Table 5 shows that, from the One-Link coverage to the Cartesian coverage, the more action-event links AEF covers, the more defects are detected and the higher code coverage is achieved. The number of GUI defects detected depends on the link-based coverage criterion used.

The Cartesian product results in the same level of code coverage as the traditional PAM approach. This result can be different on a more complicated GUI because the traditional PAM approach can explore any combination of events. However, all-event-transition coverage can be achieved only at the expense of test case explosion. AEF, in contrast, explores combinations of events that correspond to the semantics of the abstract action-level test cases. In AEF, the invocations of all GUI events corresponding to an action must finish before performing the next action of an action-level test case.

Both approaches did not detect three defects at the first execution of test cases. However, these defects are uncovered when we augment both approaches with stronger test oracles to verify more GUI attributes. For example, in Figure 4, the mapping function *deletetask* indicates that after a task is deleted, the tree control is checked to make sure that the number of tasks is reduced by one. However, one of the three uncovered defects just deleted the wrong task. In this case, the number of tasks is reduced by one, but the task to be deleted is still on the tree view while another task is mistakenly removed. This defect can be detected by strengthening the test oracle of *deletetask* in both the action and the mapping function.

Instead of checking only the number of tasks in the task list, the test oracle should include other attributes of the task list as well. We modified the action function so that it returns a *ToDoList* object. In the mapping function, that *ToDoList* object is mapped to corresponding attributes of the tree view such as the number of tasks and the names and progress of every task in the list.

The results in Table 4 and Table 5 show that, when modelling the same set of actions and events, AEF can achieve a good level of code coverage and can have reasonable defect-detection ability, while potentially saving significant modelling effort in comparison with the PAM approach.

In the next part of the case study, we studied the benefit of using the action model to test the business logic of the system under test (BL testing). This requires an action test harness which connects the actions in the action model with the business logic code of the system under test. We spent 8 hours to develop the action test harness. We tested both the business logic, using action test cases generated from the action model, and GUI interactions, using event-level test cases as presented earlier in this section. Results are shown in Table 6. Detected defects are divided into 2 groups. The first group consists of 6 defects which change the underlying business logic of To Do Manager, hence can also change the GUI behaviour. The other group consists of GUI defects which affect only the GUI attributes displayed to users.

The advantage of doing both BL testing and GUI testing in AEF is that testers do not need to wait until the development of the GUI finishes to perform testing. The BL code is usually developed and available before the development of the GUI finishes. Therefore, BL testing can start before the GUI is fully developed, resulting in the early detection of BL defects.

In this case study, AEF requires less testing effort than the traditional PAM approach while maintaining a reasonable defect-detection ability. The case study also shows that in AEF testers can do both BL and the GUI testing of the system under test separately from the same action model, resulting in the early detection of BL defects.

**Table 6 Defects detected by BL and GUI testing**

	Coverage criteria	Defects detected
AEF	One-Link Coverage	BL: 6 GUI: 3
	All-Link (1-way) Coverage	BL: 6 GUI: 7
	Cartesian coverage	BL: 6 GUI: 8
PAM	All-transition	14

## 5 Conclusions and future work

In this paper, we introduced AEF, a framework for MGT. The main components of AEF are an action model and a mapping model that maps actions to GUI events. The action model can be used to test the underlying business logic and then be reused in GUI Testing.

We described a case study based on a task manager application. The case study compares AEF and the traditional PAM approach in terms of modelling effort, code coverage, and defect-detection ability.

In the future, we aim to extend the mapping language to make it applicable to more complicated events and interaction scenarios. One such kind of events are observable events which are initiated not from the user but from the system. For example, a “new email” notification in an email client is an event invoked by the network when an email packet is observed at the email port. In this case, the event itself and event input do not depend on the user, but depend on other systems, hence require different modelling mechanisms.

Even though the case study shown in this paper provides an initial effectiveness evaluation of AEF, it is too small to prove AEF’s advantages. We plan to do further case studies on larger GUI systems to examine the effectiveness of AEF.

The generation of test data also needs to be investigated further. We will implement the AER tool as mentioned in Section 3.3. Lastly, the use of AEF in regression testing will be studied.

## 6 References

- Andrews, A., Offutt, J. and Alexander, R. T. (2005): Testing Web Applications by Modeling with FSMs. *Software and Systems Modeling*, vol. 4, 326-345.
- Campbell, C., Grieskamp, W., Nachmanson, L., Schulte, W., Tillmann, N. and Veanes, M. (2005): Model-Based Testing of Object-Oriented Reactive Systems with Spec Explorer. *Technical Report, Microsoft Research*.
- Daboczi, T., Kollar, I., Simon, G. and Megyeri, T. (2003): How To Test Graphical User Interfaces. *IEEE Instrumentation & Measurement Magazine*, vol. 6, 27-33.
- Fewster, M. and Graham, D. (1999). *Software Test Automation*, Addison-Wesley Professional.
- Finsterwalder, M. (2001): Automating Acceptance Tests for GUI Applications in an Extreme Programming Environment. *Proceedings of 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering*, 114-117.
- Ganov, S., Khurshid, S. and Perry, D. (2007): A Case for GUI Testing Using Symbolic Execution. *Testing: Academic and Industrial Conference Practice and Research Techniques*.
- Hartman, A. (accessed 20 August, 2008): *Model Based Test Generation Tools*. [http://www.agedis.de/documents/ModelBasedTestGenerationTools\\_cs.pdf](http://www.agedis.de/documents/ModelBasedTestGenerationTools_cs.pdf).
- Hetzel, B. (1988). *The Complete Guide to Software Testing*, Wiley, New York.
- Kervinen, A., Maunumaa, M., Pääkkönen, T. and Katara, M. (2006): Model-Based Testing Through a GUI. *Proceedings of Formal Approaches to Testing of Softwares Conference (FATES 2005)*.
- Li, K. and Wu, M. (2004). *Effective GUI Test Automation: Developing an Automated GUI Testing Tool*, Sybex.
- Memon, A. (2001): A Comprehensive Framework for Testing GUI. *Ph.D dissertation, Department of Computer Science, University of Pittsburgh, Pittsburgh*.
- Memon, A. (2002): GUI Testing Pitfalls and Process. *IEEE Computer Society Press*, vol. 35, 87-88.
- Memon, A., Banerjee, A. and Nagarajan, I. (2003a): GUI Ripping Reverse Engineering of Graphical User Interfaces for Testing. *Proceedings of 10th Working Conference on Reverse Engineering*, 260- 269.
- Memon, A., Soffa, M. L. and Pollack, M. E. (2003b): Coverage Criteria for GUI. *Proceedings of 8th European Software Engineering Conference*, 256-267.
- Neto, A., Subramanyan, R., Vieira, M. and Travassos, G. (2007): A Survey on Model-based Testing Approaches: A Systematic Review. *Proceedings of 22nd ACM International Conference on Automated Software Engineering (ASE 2007)*.
- Paiva, A. (2007): Towards the Integration of Visual and Formal Models for GUI Testing. *Proceedings of European Joint Conferences on Theory and Practice of Software*, 99-111.
- Perry, W. (1995). *Effective Methods for Software Testing*, Wiley, New York.
- Reza, H., Endapally, S. and Grant, E. (2007): A Model Based Approach for Testing GUI Using Hierarchical Predicate Transition Net. *Proceedings of International Conference on Information Technology*, 366-370.
- Strelzoff, A. and Petzold, L. (2003): Decision Tree Organization for GUI Generation. *Proceedings of IEEE/NASA Software Engineering Workshop*.
- Thommen, K. (accessed 20 August, 2008): To Do Manager. <http://sourceforge.net/projects/todo-manager-cs>.
- Utting, M. and Legeard, B. (2007): *Practical Model-based Testing*, Morgan Kaufmann.
- White, L. and Almezen, H. (2000): Generating Test Cases for GUI Responsibilities Using Complete Interaction Sequences. *Proceedings of International Symposium on Software Reliability Engineering*, 110-121.

# Improving End-User GUI Customization with Transclusion

Lung-Chen Lee

Christof Lutteroth

Gerald Weber

Department of Computer Science  
The University of Auckland

38 Princes Street, Auckland 1020, New Zealand

Email: llee051@aucklanduni.ac.nz, lutteroth@cs.auckland.ac.nz, gerald@cs.auckland.ac.nz

## Abstract

Usually the possibilities for end users to customize GUIs to their requirements are limited. We present a GUI specification and customization system, the Auckland Interface Model (AIM), that represents GUIs as documents that can be loaded, saved and changed by the end user during runtime. GUI layout and GUI content can be customized independently, and GUIs can be decomposed using transclusion. In this paper, we explain why transclusion is an important feature for GUI customization that does not only facilitate the maintenance of a GUI, but also supports its consistency and clarity. Transclusion makes it easier to reuse GUI specifications, and support different customization scopes. AIM was implemented on several platforms and evaluated using the cognitive dimensions framework.

*Keywords:* Transclusion, GUI customization, end-user development.

## 1 INTRODUCTION

Graphical user interfaces (GUIs) are the most common type of user interface in modern software applications. Developing a good GUI requires a significant amount of work because of usability considerations such as learnability and suitability for a particular task. It is usually hard to understand the exact requirements of users upfront. As a result, it benefits both developers and end-users if a GUI can be customized easily. Developers can use GUI customization as a rapid development tool, i.e. to react to user feedback quickly. End-users can use GUI customization to satisfy special requirements, such as simplified GUIs for novice users or larger widgets for users with visual impairments. However, most GUI applications offer only limited customization options. Making a GUI customizable usually involves significant extra work for the developers.

In this paper we describe how GUI customization can be improved by supporting transclusion of GUI specification parts. We describe a document-oriented GUI customization system (Draheim et al. 2006), the Auckland Interface Model (AIM) (Lutteroth & Weber 2008), which has been implemented on the Java, .NET, and Haiku platforms. Document orientation means that GUIs can be treated like documents, i.e. they can be modified, saved and loaded at runtime. Transclusion means that a document includes parts

of another document by reference. In AIM, transclusion can be used to decompose GUI specifications, and to use and adapt specification parts in a different context. From a developer's perspective, this facilitates reuse and maintenance. From an end-user perspective, this facilitates end-user development, consistency among GUIs, and scoping of application data.

This paper begins with a discussion of the benefits and the current limitations of GUI customization. The following section introduces the concepts and advantages of the document-orient approach as well as its implementation in the Auckland Interface Model, followed by a discussion of how document decomposition with transclusion improves the maintenance and the clarity of GUIs. Finally, our approach is evaluated with the Cognitive Dimensions framework.

## 2 RELATED WORK

### 2.1 Findings on End-User Customization

Different users may have different requirements for the user interface of an application. Although a GUI designer may capture the common requirements for most users, it may satisfy users' individual needs much better by allowing the users to customize the application. A study by Findlater et al. (Findlater & McGrenere 2004) showed that the users were able to customize the user interface to increase the efficiency. The majority of users also preferred a personalized interface. A later study (McGrenere et al. 2007) on user's customization on a word processing software with two types of menu interfaces also shows the users do customize effectively when they are provided with an easy to use customization mechanism. The users prefer the adaptable menu system which they can control over the system-controlled adaptive menu system.

Page et al. (Page et al. 1996) presented a study on the how users customize a word processor. The study finds that 92% of the participants customize the software in some aspects. It indicates that users do customize software applications when they are presented with the options. The results show that the users tend to customize the application more if they use the software more. The work they surveyed suggested that the common obstacles to customizing software were the lack of time and the difficulty to customize the application. The study recommends a mindset of expecting the users to customize the software. The customization should be made easy for them to use. The users should be able to switch between normal task and the customization activities with little disruption.

In the discussion about the past, present and future of user interface software tools presented by Myers et al. (Myers et al. 2000), it was suggested that a "gentle slope system" that the user only needs to learn incrementally would attract more users to overcome

the initial threshold of customization. It also proposed that these capabilities should be supported by the underlying toolkit instead of by the individual applications. This is in line with the Auckland Interface Model, which provides the customization capabilities to all applications that use this system.

The studies above indicate that the users indeed customize the user interfaces of the applications and it helps their efficiency and satisfaction of using the software application. However, at present, many applications provide the end-users a small degree of GUI customization. The customization is usually limited and may not persist between sessions or it may have troubles in a multi-user environment (Kim & Lutteroth 2009). Providing graphical user interface customization functionalities in the application requires the developers to spend a significant amount of effort.

The traditional approach of developing the GUI of a software application is to include the GUI as code in the program, which is mixed with the code for other functional parts of the program. There are some limitations with this approach. It is more difficult to develop and maintain the program logic and the user interface independently. It may cause difficulties for the programmer and the user interface designer to work on their parts separately. It also makes it hard to customize a user interface. As the user interface is represented as code, the customization requires changing the code and recompiling the program. Furthermore, because the user interface is described in a general purpose programming language, it is possible that the user interface is generated by a complicated section of code. It is difficult for the developers to maintain and modify as it is hard to statically analyze the GUIs by only reading the code without executing the program.

## 2.2 End-User Customization Technology

Systems with various aims have been developed to enhance the editing of the GUI and they have varying effects on GUI customization for the end-users. Existing approaches for GUI customization work either in the windowing system, or rely on special widget toolkits.

Implementing customization features into the windowing system has the advantage that all applications can be customized. However, customization is very generic and does not take into account the exact structure of a GUI. For example, User Interface Façades (Stuerzlinger et al. 2006) is a system that allows the user to copy and paste screen regions to duplicate the GUI controls and arrange them for their convenient use.

Using special widget toolkits drastically restricts the number of applications that can be customized, but customization features can be closely integrated with the structure and logic of a GUI. For example, EUPHORIA (McCartney et al. 1995) is a user interface management system that allows the end-user to construct the user interface of distributed multimedia applications with custom widgets.

## 2.3 GUI Description Languages

The main benefit of using separate languages to specify GUIs is the separation of concerns (Draheim et al. 2006). Separating the specification of the GUI from the program logic makes the structure of an application clearer, and allows developers to edit the GUI without navigating the program logic. As such, GUI description languages are a software engineering technique more than an interaction principle.

Many GUI description languages have been designed to specify GUIs declaratively. Two better

known examples are Mozilla XUL (Cheng 1999) and Microsoft XAML (Bishop 2006). There are other user interface languages such as UIML (Abrams et al. 1999) that have been developed but never gained broad acceptance. All these approaches use documents – typically XML – to specify a GUI, therefore we call them *document-based*.

In principle a document-based GUI is amenable for end-user development, if end-user development is understood as development through a highly skilled power user. It is possible for power end-users to edit the documents of GUI descriptions to change the GUI. In the case of the code-based GUI, the customization may involve changing the code and recompilation, which even a power end-user typically cannot do. The focus of the current GUI specification technologies is on improving the GUI construction and maintenance for UI designers and programmers.

## 2.4 Transclusion

The general concept of transclusion is to include one document into another document by reference. In this discussion, the document that includes another document is referred to as the transcluding document or the transcluding page. The document that is included into another document is referred to as the transcluded document or the transcluded page.

The term “transclusion” was coined by Ted Nelson. He proposed a comprehensive system of viewing and sharing digital documents, known as Project Xanadu (Nelson 1981, 2007). Transclusion specifies how documents are stored, addressed and edited to construct different versions. It emphasizes that the connection between the transcluding document and the transcluded document must be maintained.

Ted Nelson’s idea is that of a “docuverse” that enhances the viewing, editing and sharing of digital documents more than merely imitating paper (Simpson et al. 1996, Nelson et al. 2007). It provides a visualization of documents that indicates the connection between the quoted content and the original context, and allows readers to view the original context. Content is reused by referencing the original source without explicit copying of the content (Nelson 1995). A document is made of a sequence of global reference pointers that point to segments of content (Nelson n.d.). Changing a document is regarded as creating a new version of the document and is done by editing the list of reference pointers to point to a new sequence of segments. The characters at the existing content addresses never change.

Later approaches by other research groups adopt the main feature of maintaining the connection between a transcluding and a transcluded document, but support only some of the features of Nelson’s original transclusion idea. For example, Di Iorio and Lumley investigated adding transclusion to XML (Di Iorio & Lumley 2009). It attempts to add an transclusion mechanism to XML without building an entirely new system like Project Xanadu. Another example is Krottmaier proposed a system to perform transclusion for LaTeX documents to enable the inclusion of content by reference (Krottmaier 2002). The transcluding document embeds a macro, and a server processes the macro to compose the document with the transcluded content before sending it to a client.

## 3 DOCUMENT-ORIENTED GUIs

The document-oriented approach views a GUI as a structured document analogous to forms that can be created in office applications (Draheim et al. 2006).

Many capabilities of the GUI can be explained in terms of this metaphor. For example, the ability of the user to interact with many typical controls such as text areas and check boxes is analogous to filling out a predefined form.

Document-orientation means that the UI rendering infrastructure incorporates editing functionality, and therefore GUIs can be edited directly by end-users. The GUI framework behaves like a WYSIWYG office application. Since customization will typically be viewed as a separate activity from actually using the GUI, the GUI framework distinguishes an edit mode, where the GUI can be directly edited, and an operational mode. The developers can give editing access rights to end-users, and hence control the level of customization that is allowed.

In AIM, our implementation of a document-oriented system, the customization function is integrated into a layout manager. As a result, customization is available in every GUI that uses the layout manager, without additional development effort. After switching to the edit mode, parts of the GUI can be visually rearranged, removed or replaced. This is supported by direct manipulation, e.g. users can drag and drop parts of the GUI. More information about the edit mode of AIM can be found in (Lutteroth & Weber 2008).

AIM supports a particularly strong separation of concerns in GUI applications. Whereas other technologies support only a separation of user interface and application logic, AIM further separates parts of the GUI: it is possible to separate the GUI layout from the GUI content and the GUI data. GUI content refers to the widgets in a GUI, and GUI data to the data values that are represented in the widgets. The different concerns of GUI specification in AIM are described in the following.

### 3.1 GUI Layout

The Auckland Layout Model (ALM) is a mathematical model that allows precise, platform independent specification of GUI layouts, i.e. the visual arrangement of widgets in a GUI, and their dynamic behavior, i.e. how they change under resizing. The layout of a GUI is described by specifying horizontal and vertical tabstops – grid lines that are used to align widgets – and linear constraints. Each widget area is bound by a pair of x-tabstops and a pair of y-tabstops. ALM uses linear programming to compute the layout of a GUI.

GUI layouts can be loaded and saved in a platform-independent specification language. This language refers to the widgets in a layout by symbolic names. This means that the layout can be specified independently from the widgets and their content.

### 3.2 GUI Content

The GUI content of AIM is handled by the Auckland Value Model (AVM). AVM provides a description language to specify the GUI content, i.e. all the widgets in a GUI and their properties. AVM is platform independent, i.e. it defines abstract widget types. When an AVM specification is loaded, platform-specific adapters are used to instantiate and configure widgets so that they match the specification. AVM supports loading as well as saving.

### 3.3 GUI Data

AIM also enables the data that are presented in the widgets of a GUI to be customized. For example, such data could be strings in textboxes, selected items in

lists, check boxes or radio button groups, or numerical values in sliders. The data in such widgets are stored as widget properties in an AVM description. By loading and saving AVM specifications, users can manage, reuse and share the data represented in a GUI.

## 4 DECOMPOSITION OF GUIs

Document-orientation supports the decomposition of GUIs, similar to the way it is done for other types of documents. GUI specifications in AIM are composed of parts such as layout areas and constraints in ALM layout specifications, and widgets with properties and data values in AVM content and data specifications. In order to reuse GUI specification parts, transclusion is employed.

When a change is made to a part, the change is reflected in all documents using the part. This is similar to the way a change in a style sheet affects all the documents that use the style sheet. It enables centralized maintenance, while still enabling special customizations in individual user documents. If transclusion is not used and the same part is duplicated in multiple documents, changes would have to be made to all the individual documents, which would frequently lead to inconsistencies. The following sections discuss advantages and usage scenarios of GUI decomposition using transclusion.

### 4.1 Sharing and Reuse

AIM documents can easily be copied and shared. The capability to share and reuse a customized GUI is a useful feature for end-users as well as for developers. For example, if one person in an office customizes the GUI appearance and the program settings of an application to be convenient for the tasks performed in that person's team, it would be great for other team members to have the same adjustments to their GUI. Without document orientation, the person would have to repeat the customization steps on each of the team member's computers. Naturally, it is much easier if the customized GUI can directly be shared with the other team members. Team members benefit from the customization, and retain the flexibility to perform individual adjustments. For developers the concept of reuse is very common, and in fact an essential principle of software engineering. Hence, having a reuse mechanism for GUI parts makes GUI development easier.

### 4.2 Consistency

A main application of GUI transclusion is to ensure consistency in GUIs. This may be consistency in the layout, the widget appearances, or the GUI data.

#### 4.2.1 Consistent GUI Appearance

When GUI designers develop a GUI, they usually want it to look coherent overall. They are likely to design similar widgets so that they have a common style. For example, buttons should typically have the same background color, and textboxes should use the same font. This is similar to the use of style sheets as commonly used in word processing documents and web user interfaces.

In AIM, designers may create template widgets for a GUI, e.g. a template button with a particular color and font. If the buttons on the GUI all transclude the template button, they have the same appearance as default because the buttons reuse the properties of the template button. When the UI designers or the users

change the template button, all buttons change to the new appearance. As the designers frequently edit the user interface to try out new styles, it is easy to make a global change. It is less likely that individual widgets are forgotten and end up having inconsistent appearances.

A similar approach can be taken with GUI layouts. Sometimes features of a layout are reused. For example, many layouts have toolbars at the top and a status bar at the bottom. By reusing layout parts, consistency can be achieved more easily. For example, a GUI designer may decide to locate the toolbars at the side instead of the top. With the proper use of transclusion, such a change can automatically adjust all the GUI windows of an application suite accordingly.

#### 4.2.2 Consistent Program Settings

Transclusion of GUI data is useful for options and preferences dialogue windows, where frequently different sets of settings need to be managed. For example, a user may want to use the same print dialogue settings in a web browser, a word processor and a PDF document viewer. Using AIM, different applications can use the same data for the settings in a dialogue.

The settings are stored as properties of widgets in an AVM description. The user can set an application to use the same AVM description for a part of the GUI as another application. Consequently, different applications can present the same print settings to the user.

AIM can also be used to manage different sets of data that are used in different contexts. For example, if the user needs different settings for printing photos than for printing presentation slides, the user may load different AVM descriptions of the print dialogue with different values of the GUI widgets.

Managing program settings in the GUI is also useful for system administrators. If the users' GUIs transclude the settings from the system wide setting, all users would have the same settings by default. When a system administrator changes a particular system-wide setting, all users get the new default setting. It avoids the possible inconsistencies when settings are changed individually for each user. An administrator can also use this mechanism to manage different settings for different groups of users.

#### 4.3 Overriding

Transclusion allows different users to reuse the same GUI parts. In AIM, users can also customize the transcluded parts without breaking the relationship between the transcluder and the transclusee. Individual properties of a transcluded part can be overridden in the transcluding document. For example, a user may transclude all the system-wide print settings, but override the default paper size. The user may later discard their individual program settings to use the system-wide settings again. To do this, the user only needs to remove the personal settings that override the system-wide settings, i.e. the transclusion from the system-wide settings will still work.

In comparison, Cascading Style Sheets (CSS), which is used for web user interfaces but not for standalone applications, also offers overriding of UI appearance to a limited extent. The difference is CSS does not allow as fine-grained customization as AIM does. CSS style sheets can only import other style sheets as a whole; it is not possible to import only parts of another style sheet, whereas AIM specifications can transclude individual widgets selectively.

#### 4.4 Scoping

Document-orientation makes it possible to customize GUIs in different scopes, such as between sessions or across different applications. For example, an application may hold the only copy of a particular print dialog window specification. The same dialog window is displayed to all users because the same GUI specification is used application-wide. GUI declaration units can also be specified in user-specific documents, thus allowing different users to have different dialog windows. Separate copies of GUI specifications for the same dialog window enable individual users to have customized user interfaces for the same application, while some parts of a GUI can stay the same for all users. Typical scopes would be user, organization, application, platform, session, and document.

The decomposition mechanism allows different scopes of customization to suit different roles. For example, the system administrator is able to change some settings on an application, and the change takes effect for all users when they use the application. Individual users may customize their own UI to suit their individual needs. The decomposed GUI declaration units are stored at appropriate locations to enable the different roles to access them.

For example, an IT administrator may supply default print settings to all users. As AIM supports different scopes of customization, when the IT administrator wants to change the default values for all users, she can edit the values contained in the AVM description of the print dialogue. The change affects all users. If an individual user wants settings different from the default settings, the user can edit the AVM description that is used only by the user.

Not all document concepts support fully recursive transclusion. HTML notoriously did not do so from the start. In contrast, in our implementation of the document-oriented approach, we support full transclusion, as well as standard document locations. This allows the user or application designer to store information per application, per user or per file. The concept of the scope of a GUI customization is thus mapped to the concept of a file location, a good example, how the document metaphor is mapping the semantics of the customization to familiar concepts.

#### 4.5 Scope Visualization

This approach also motivates a refinement of transclusion: the location of the data is not hidden from the user, as it would happen in most other approaches, but actually displayed to the user; moreover it can be changed by the user. Since this location information is chiefly logical and non-obfuscated (i.e. it says "scope: user bert" or "scope: application myEdit-Pad" instead of a URL or an equally opaque UNIX file path) it enhances documentation and transparency of the user interface. This is in stark contrast to many auxiliary documents in standard office applications, where the scope of a setting is very unpredictable, sometimes being the file, sometimes the office suite, sometimes the session, and this scope is not made explicit to the user, let alone being configurable. Finally, the fact that we see GUIs as documents editable by a WYSIWYG editor gives the user full control over GUIs.

In our design, using transclusion enables the system to provide the information about the scope of setting the value of a component on the interface. An example is for changing the settings in the print dialogues of various software applications. Some applications like the Microsoft Office applications use the same print dialogue for all open instances of the same application. If the users change the option to turn



on double-sided printing on one MS Word window, and then switch to another instance of Word to print, the second instance also has the double-sided printing turned on. Other applications may behave differently when print settings are changed. For example, OpenOffice.org Writer is another word processing application. When the print settings are changed, the changes only affect this instance of the application. Print settings in other instances of OpenOffice.org Writer are not affected. The inconsistent scope of changes in different applications may confuse the users.

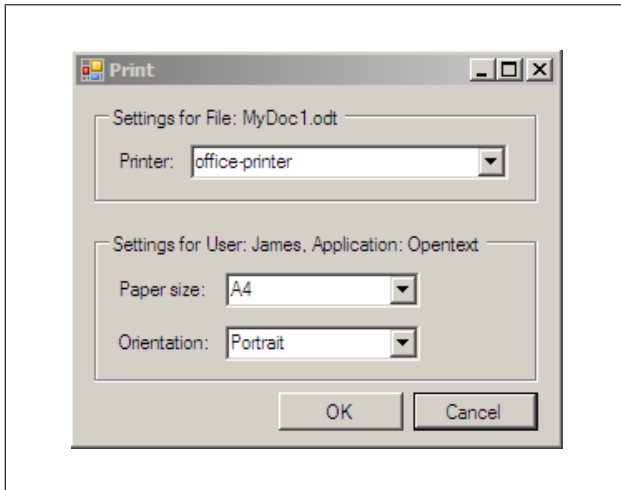


Figure 1: A print dialogue consists of options of different scopes, which are indicated by the descriptions of the group boxes

When transclusion is used, the UIs of the settings are loaded from different settings files, which contain the values held by the components. The resulting UI may be composed of components from different scopes. An example of a print dialogue is shown in Figure 1. The system knows which GUI control is from which UI document file and can use this information to inform the users about the different scopes of the controls. Extra visual elements are introduced in the GUI to convey this information.

One possible way to indicate the scopes of the different GUI controls is to decorate controls individually. Another way is to use visual containers to separate the controls in different groups. A visual container encloses the UI components that are concerned with settings that affect only this particular file, and another container encloses controls that belong to another scope. In the example shown in Figure 1, a group box is used to enclose the components. The caption of the container indicates the scope of the settings that are inside this container. In this example, the Printer setting is for this particular file, MyDoc1.odt. If the user edits this setting, it does not affect the same print dialog setting used in other files.

The example has another container that contains the settings for the particular user's files that are opened by this application. The settings 'Paper size' and 'Orientation' are inside this container. If the user James changes the paper size from A4 to A3, the other files James opens using the same application Opentext will also have the paper size as A3 when he opens the print dialogue. The extra visual elements clarify which settings have which scopes of effect. The users are better informed when they use this GUI and they are more confident that the changes they make will not cause unintentional changes.

The user interface also allows the user to change

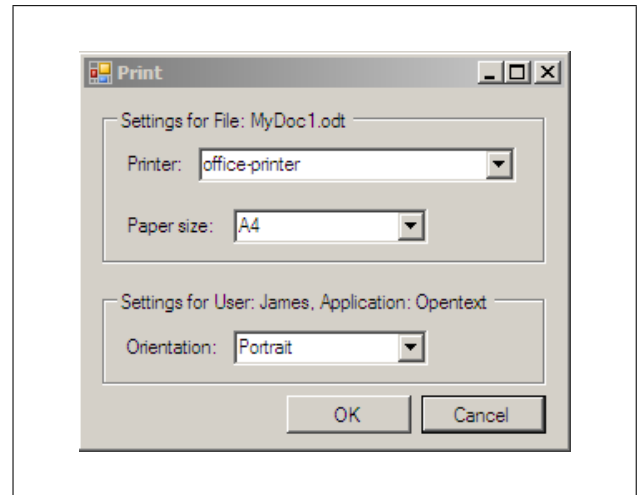


Figure 2: A print dialogue with the scope of the paper size setting as file-specific

the scope of GUI controls when the GUI is in the editing mode. There are several proposed ways of changing the scopes and they will be investigated in our research. As AIM allows the users to move the GUI controls when it is in the editing mode, one possible way to change the scope of a GUI control is to simply move it to inside of the container of the desired scope. Another possibility is to drag and drop the edge of the group container to include new widgets. For example, Figure 2 shows that the scope container for the file has been expanded to include the paper size setting as a setting for only this file.

## 5 IMPLEMENTATION

The GUI content of AIM is handled by the Auckland Value Model (AVM) component. AVM provides a content description language to specify the GUI widgets and GUI values, i.e. all the widgets in a GUI and the values they hold.

An engine called the AVM Engine processes the content description language. It facilitates instantiating widgets from the content specification, and saving widgets to the content description. A current implementation of the AVM Engine is written in the Java programming language. It converts widgets between the AVM description and the Java Swing GUI components. It also handles the transclusion of the widgets.

### 5.1 AVM Specification

AVM uses PDStore as its data access layer and it defines the model of the AVM specification. PDStore is a structured database system that uses GUIDs (globally unique identifiers) for all data instances. An association between instances also has a GUID, which is the role ID that is used to find the associated instances of an instance. An example in the AVM specification is that a button has a role 'background'. We may use the ID of the 'background' role to access the instance in PDStore that represents the background value of the button. PDStore allows the AVM Engine to access the AVM specification stored in the database in a more object-oriented manner.

An AVM specification is defined as having a number of widgets. A widget has a value, which represents the value held by a widget such as the text of a text box. It also has a transclusion associated with it to specify which widget it transcludes. A widget is associated with an instance of a particular type of widget, e.g. a text box or a button. This concrete

type of widget has a number of property values. This is summarized with the following diagrams. Figure 3 is a trimmed down model of the AVM specification showing two types of widgets each with two properties. Figure 4 is a sample of two instances of AVM specifications.

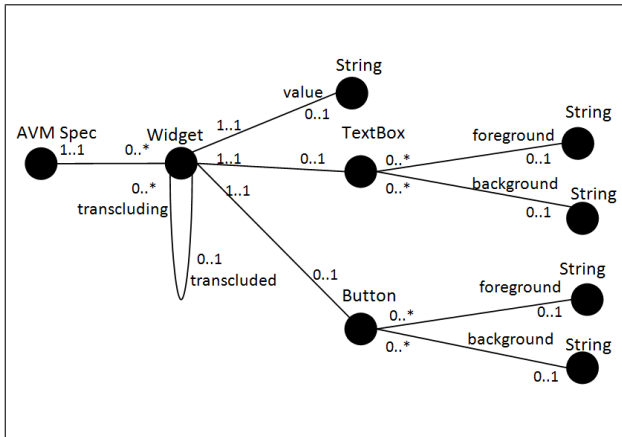


Figure 3: PD model of AVM specifications.

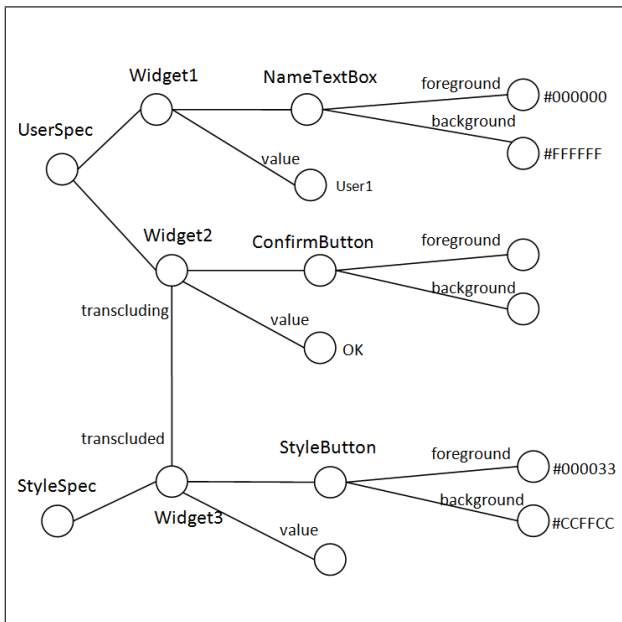


Figure 4: Two AVM specifications. Widget2 transcludes Widget3.

## 5.2 Transcluding Widget Values

A widget may hold its own value, in which case this value is used. If the value of a widget is null, the widget uses the value of the widget it transcludes. If the transcluded widget does not hold a value, either, it in turn finds its transcluded widget and uses the value of that widget. The search ends when it finds a value.

The following shows an example of the text box that holds the value of the paper size in a print dialogue. The text box in the user's AVM specification transcludes the paper size text box in the department-wide AVM specification of the print dialogue. It in turn transcludes the size text box in the faculty-wide AVM specification. When the AVM Engine loads the paper size text box, it finds the its value is null. So it looks for the transcluded department-wide widget, which does not contain a value, either. It then finds

the transcluded faculty-wide widget, which contains the value A4. In the end, the AVM Engine uses A4 in the paper size text box for the user's print dialogue.

A case could be the department finds that printing on the larger A3 paper is the norm for this department, and decides to change the default paper size for all users in the department. It then replaces its the null value of the paper size widget to A3. The AVM Engine looks for the value in the transcluded widget, it finds A3 and uses it.

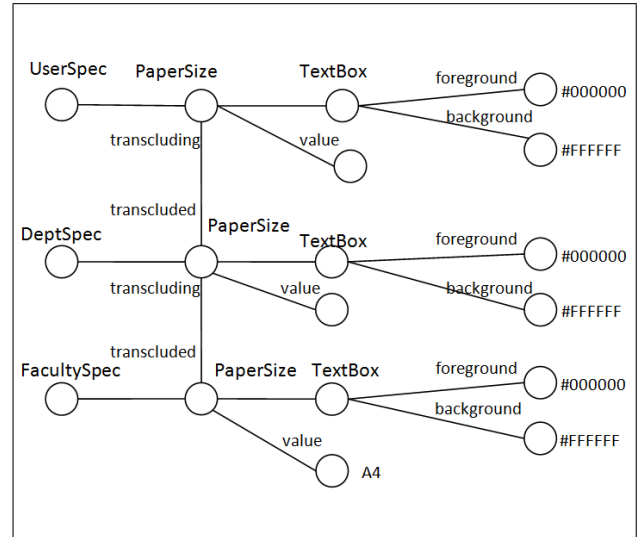


Figure 5: The widget PaperSize in UserSpec transcludes the value 'A4' from PaperSize in FacultySpec.

The following pseudo-code describes loading of transcluded widget values:

- 1: **while** value = null and current widget  $\neq$  null **do**
- 2:   Read the value of the current widget
- 3:   Find the transcluded widget and make it the current widget for subsequent processing
- 4: **end while**

## 5.3 Transcluding Widget Properties

The widget properties, which determine the appearance of the widget, may also use the property values of the transcluded widget. For example, a GUI designer makes a button of light green background and dark blue text and wants it to be the default style of the buttons of the GUI. The designer edits all the buttons to transclude this button and leaves all property values as null. Figure 4 is an example. When the AVM Engines loads a button, it looks for the property values. It records all properties that have values. There is none in this case. It then finds the transcluded widget. It checks the widget is of the same type, i.e. a button. The AVM Engine then read property values for properties that do not have values yet. In this case, it records all the property values of the template button. As it has collected non-null values for all properties, the search for transcluded property values is complete. The AVM Engine uses the collected property values to set the appearance of the instantiated widget. For this example, the button would have the same appearance as the template button.

The following pseudo-code describes loading of transcluded widget properties:

- 1: Get the set of widget property role IDs of this type of widget from the widget adapter
- 2: **while** set of property role IDs is not empty and the current widget is not null **do**

```

3:   for each property that has not obtained a
    value do
4:       Use the property role ID to access the
    property value of the current widget
5:       if the value is not null then
6:           Record (role ID, value) in the hash
    table for property values
7:           Remove the role ID from the set of
    property role IDs
8:       end if
9:   end for
10:  Find the transcluded widget and make it the
    current widget for subsequent processing
11: end while
12: for each property remaining in the set of property
    role IDs do
13:     Set the property to its system default value
14: end for
15: for each property in the hash table do
16:     Retrieve its value and use the widget adapter
    to set the corresponding property of the instanti-
    ated widget
17: end for

```

Suppose the designer wants a few buttons for a particular section of the GUI to have a bright yellow background. The designer creates a new widget of the button type in the AVM specification. The new button has yellow as its background color and holds null for all other property values. It transcludes the template button. The designer edits the buttons of the particular section to transclude the new button. When the AVM Engine loads a button, it again does not find any property values in the button itself. It goes to the transcluded widget and finds the newly created button holds the value of the background color. It records yellow as the background color and goes to the transcluded widget to find other property values. It records the values of other properties of the template button. The AVM Engine uses the property values to instantiate a button that has the appearance of the template button except the background color is yellow. The text is dark blue as it uses foreground value of the template button but it has yellow as the background value comes from the new button. The AVM specification is shown in Figure 6.

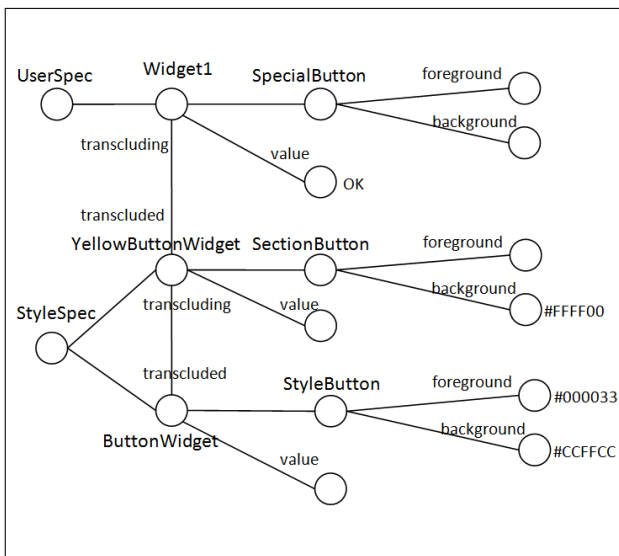


Figure 6: Widget1 uses the background color from YellowButtonWidget and uses the foreground color from ButtonWidget.

## 5.4 Changing Widget Values

When the user uses a new value for a widget that currently uses a transcluded value, the widget replaces the null value with the new value. The widget uses its own value and properties it holds, while using transcluded values for other properties. Effectively, the widgets' own values override the transcluded values. If the user later decides to change back to using the transcluded value, it is done by changing the value to null to specify the value is to be found in the transcluded widget.

## 5.5 Widget Adapters

The AVM Engine uses a set of widget adapter classes to process the various types of widgets. A widget adapter provides a standard API that allows AVM Engine to access the properties of a particular type of widgets. The API wraps the actual platform specific calls that access the widget properties. Therefore, the AVM Engine is not concerned with platform specific details. New widget adapters can also be easily added to the AVM Engine to provide the processing capability for a new type of widgets.

A widget adapter provides a standard API that allows the AVM Engine to access the properties of a particular type of widgets. The API of the widget adapter is as the following:

```

1  public interface WidgetAdapter {
2      JComponent New();
3      GUID getTypeId();
4      GUID getWidgetRoleId();
5
6      Object getValue(JComponent widget);
7      void setValue(JComponent widget,
8                  Object value);
9
10     GUID[] getAllPropertyRoleIds();
11     String getPropertyType(
12         GUID propertyGUID);
13     Object getProperty(JComponent widget,
14         GUID propertyGUID);
15     void setProperty(JComponent widget,
16         GUID propertyGUID, Object value);
17
18     Object getPropertyDefaultValue(
19         GUID propertyGUID);
20     boolean hasPropertyDefaultValue(
21         JComponent widget, GUID propertyGUID);
22     void resetPropertyToDefaultValue(
23         JComponent widget, GUID propertyGUID);
24 }

```

The method New is used for instantiating an actual widget in the program. The type of widget adapter determines the type of widget that is created. The getTypeId and getWidgetRoleId methods are used to tell the AVM Engine the type ID and role ID of this type of widget in the PD model. It enables the AVM Engine to retrieve the corresponding instance of this type of widget in the AVM specification.

getValue and setValue provide access to the property that is considered as the value of a particular type of widget. For example, the text inside the text box is considered as the value of a text box.

getAllPropertyRoleIds tells the AVM Engine all the properties in a type of widget. getPropertyType indicates the value type of a property, so the AVM Engine knows how to parse the property value. The property value is accessed by using getProperty and setProperty.

For a property, `getPropertyDefaultValue`, `hasPropertyDefaultValue`, and `resetPropertyToDefaultValue` allow the AVM Engine to retrieve the default value, check whether a widget holds the default value, and set it to the default value for the widget respectively.

The API encapsulates the platform specific calls that access the widget properties. Internally, the widget adapter translates the standard call from the AVM Engine to an actual method call on a GUI component instance in the underlying GUI platform. For example, the text of a text box is considered the value of the text box widget. When setting the text of a text box, the AVM Engine calls: `adapter.setValue(nameTextBox, "Robert");` The `TextBoxAdapter` then translates it into: `nameTextBox.setText("Robert");`

New widget adapters can be developed by implementing the API. The new widget adapters can be added to the AVM Engine to enable it to process more types of widgets.

## 6 EVALUATION

We used the Cognitive Dimensions framework (Green & Petre 1996) to evaluate the use of the transclusion mechanism within the document-oriented paradigm. This framework allows us to qualitatively evaluate our approach against a set of standard dimensions, and to find issues with the approach. This framework has been used successfully in the past (Reid & Plimmer 2008). We used the print-dialogue and the page settings dialogue in text processing applications, and compared their implementation in standard text processing applications with a mock-up implementation based on AIM. Dimensions that are not yet addressed are: progressive evaluation, viscosity, visibility and role-expressiveness.

### 6.0.1 Abstraction Gradient

Abstraction is a central feature of the transclusion mechanism. Through transclusion, sets of parameter settings can be reused, for example as default settings. Through this reuse, the individual places where this reuse happens can be oblivious to the details of those parameter settings.

### 6.0.2 Closeness of Mapping

The transclusion mechanism is a direct and immediately understandable representation of practices that can be found in every workplace even beyond the IT world. The idea that a certain assumption is by default valid for a large social context, and local deviating rules are explicitly stated, is a widespread notion. A typical example would be speed rules in traffic. There are rules by default, and these rules can be overridden by local changes.

### 6.0.3 Consistency

The central consistency question is the scope of changes to parameters. In conventional GUIs, as outlined by the examples, the scope of changes is often not explained and moreover counterintuitive. In our approach, once a number of individual scopes have been encountered, such as "settings for department", "settings for file myfile.txt", the user will understand easily notions such as "settings for user Pat".

### 6.0.4 Diffuseness / terseness

The transclusion concept presented here needs very little symbols, as is evident from the examples. The only symbol necessary is a group box, and the syntax of the context, consisting of the colon separating the context type and the context instance (user: Pat) and the comma separating different context specifications.

The standard context types which will appear in most cases, such as user, organization, and perhaps file, have a middle position between being elements of the notation itself, or being just applications of the notation, albeit in widespread use.

### 6.0.5 Error-proneness

The transclusion approach cannot prevent all errors, but it is not specifically error prone. The main errors that can be ascribed specifically to the transclusion approach are errors in the chosen context. A context too wide might create unwanted effects. This is usually prevented by access rights, i.e. the settings for the organization context can be used, but not changed by all users.

### 6.0.6 Hard mental operations

The hard mental operations are not on the notational level. On the semantic level, it sometimes needs some thought where from to transclude a setting, but it is our impression that this is a direct consequence of the desired outcome. The lack of flexibility of current systems however, where one simply has no choice, seems hardly to be an advantage.

### 6.0.7 Hidden dependencies

Through the context path, the transclusion approach clarifies dependencies that are normally hidden in GUIs.

### 6.0.8 Juxtaposability

An element of juxtaposability is planned for later versions. It should then be possible to see for one value that is overridden both, the current overriding value as well as the overridden value.

### 6.0.9 Premature commitment

In the normal editing process, if a transclusion path is changed and a value is overridden with an overriding value, the overridden value cannot be changed at this position. Another timing issue that is in connection with error-proneness is related to the scope of change. We consider the print dialogue and assume the paper size has currently the transclusion context "file". Now we want to do two changes: change the paper size and change the transclusion context to "user". Here a timing issue can arise. If one first changes the paper size and then changes the context, then this is not equivalent to first changing the context and then changing the paper size. In the first case we change a setting in a context and then immediately leave that context, thus the changed setting is not used. Intuitively this is likely an error. One of the features that is planned but not yet implemented is a "did you mean" warning in this case. If the user actually meant to do it the other way round and do the change in the new context, the system offers the possibility to bring things in order.

### 6.0.10 Secondary notation and escape from formalism

Certain mechanisms of the transclusion approach cannot be replaced by informal notions, since they are immediately evaluated in the running system. The transclusion path itself has to refer to an existing context or a new context has to be created. Secondary notation can take several forms in GUIs, and some of them, such as help systems, are beyond this presentation. The notation can in principle be enriched, for example with icons for contexts, however this requires changes to the framework.

A further secondary notation that is of interest here and that is readily available in the framework is textual comment by the individual user, directly in the user interface. This is certainly not possible in conventional GUIs. In AIM this is right away possible. Every AIM GUI comes with an inbuilt editor, which allows end users for example to add labels. The user can annotate for example the paper size input field with comments on important usages. These comments again are stored in transcluded contexts and this determines the visibility of the informal comment. Therefore AIM acts additionally as a tool support for the dissemination of secondary notation, without losing informality.

Informal content is currently boxed, i.e. the comment takes the form of labels (the borders are however invisible) that are placed at certain points. This hints at further work, where more general annotations with a transparent pane would be thinkable.

### 6.1 Summary of Evaluation

The evaluation through the cognitive dimensions framework allows us to attribute features of the transclusion approach to standard dimensions. With this method we have found issues where further work is warranted, namely in the dimensions of juxtaposability, secondary notation and premature commitment. While the first two dimensions have hinted us at interesting nice-to-haves for further work, the last dimension, premature commitment showed us a more important issue that should be addressed with more urgency.

## 7 CONCLUSIONS

Current GUI customization technologies are limited and do not take advantage of decomposition mechanisms such as transclusion. The Auckland Interface Model (AIM) is an open-source, cross-platform technology that uses the document orientation to provide easier and more comprehensive customization for end-users. AIM separates the concerns of GUI layout, GUI content and GUI data, and enables end-users to treat GUIs as documents. This makes it possible to customize GUIs and decompose them using transclusion. The use of transclusion has advantages for GUI developers and end users: it facilitates reuse, consistency, introduces different scopes of customization, and clarifies to the end user where values in a GUI are actually stored. AIM was evaluated using the cognitive dimensions framework, and we could identify the dimension *premature commitment*, for instance, as a potential issue that should be addressed in further work.

## References

Abrams, M., Phanouriou, C., Batongbacal, A., Williams, S. & Shuster, J. (1999), 'UIML: An

appliance-independent XML user interface language', *Computer Networks* **31**(11), 1695–1708.

Bishop, J. (2006), Multi-platform user interface construction: a challenge for software engineering-in-the-small, in 'ICSE '06: Proceedings of the 28th international conference on Software engineering', ACM, pp. 751–760.

Cheng, T. (1999), XUL – creating localizable XML GUIs, in 'Proceeding of the 15th International Unicode Conference'.

Di Iorio, A. & Lumley, J. (2009), From xml inclusions to xml transclusions, in 'HT '09: Proceedings of the 20th ACM conference on Hypertext and hypermedia', ACM, New York, NY, USA, pp. 147–156.

Draheim, D., Lutteroth, C. & Weber, G. (2006), Graphical user interfaces as documents, in 'Proceedings of CHINZ 2006 – 7th International Conference of the ACM's Special Interest Group on Computer-Human Interaction', ACM Press.

Findlater, L. & McGrenere, J. (2004), A comparison of static, adaptive, and adaptable menus, in 'CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems', ACM, pp. 89–96.

Green, T. R. G. & Petre, M. (1996), 'Usability analysis of visual programming environments: A 'cognitive dimensions' framework', *Journal of Visual Languages and Computing* **7**(2), 131–174.

Kim, J. & Lutteroth, C. (2009), Multi-platform document-oriented guis, in 'AUIC '09: Proceedings of the 10th Australasian User Interface Conference', ACM.

Krottmaier, H. (2002), Transcluded documents: Advantages of reusing document fragments, in 'ELPUB'.

Lutteroth, C. & Weber, G. (2008), End-user GUI customization, in 'Proceedings of CHINZ 2008 - 9th International Conference of the ACM's Special Interest Group on Computer-Human Interaction', ACM Press.

McCartney, T., Goldman, K. & Saff, D. (1995), 'EUPHORIA: End-user construction of direct manipulation user interfaces for distributed applications', *Software - Concepts and Tools* **16**(4), 147–159.

McGrenere, J., Baecker, R. M. & Booth, K. S. (2007), 'A field evaluation of an adaptable two-interface design for feature-rich software', *ACM Trans. Comput.-Hum. Interact.* **14**(1), 3.

Myers, B., Hudson, S. E. & Pausch, R. (2000), 'Past, present, and future of user interface software tools', *ACM Trans. Comput.-Hum. Interact.* **7**(1), 3–28.

Nelson, T. H. (1981), *Literary Machines*, Mindful Press.

Nelson, T. H. (1995), 'The heart of connection: hypermedia unified by transclusion', *Commun. ACM* **38**(8), 31–33.

Nelson, T. H. (2007), 'Transliteration: A humanist format for re-usable documents and media'. <http://transliteration.org/>.

Nelson, T. H. (n.d.), 'Xanalogical structure, needed now more than ever: parallel documents, deep links to content, deep versioning, and deep re-use', *ACM Comput. Surv.* p. 33.

- Nelson, T. H., Smith, R. A. & Mallicoat, M. (2007), Back to the future: hypertext the way it used to be, *in* 'HT '07: Proceedings of the eighteenth conference on Hypertext and hypermedia', ACM, pp. 227–228.
- Page, S. R., Johnsgard, T. J., Albert, U. & Allen, C. D. (1996), User customization of a word processor, *in* 'CHI '96: Proceedings of the SIGCHI conference on Human factors in computing systems', ACM, pp. 340–346.
- Reid, P. & Plimmer, B. (2008), A collaborative multimodal handwriting training environment for visually impaired students, *in* N. J. Bidwell, ed., 'OZCHI', Vol. 287 of *ACM International Conference Proceeding Series*, ACM, pp. 195–202.
- Simpson, R., Renear, A., Mylonas, E. & van Dam, A. (1996), '50 years after "as we may think": the brown/mit vannevar bush symposium', *interactions* **3**(2), 47–67.
- Stuerzlinger, W., Chapuis, O., Phillips, D. & Rousel, N. (2006), User interface façades: towards fully adaptable user interfaces, *in* 'UIST '06: Proceedings of the 19th annual ACM symposium on User interface software and technology', ACM, pp. 309–318.



## Author Index

Arif, Mohammed Jubaer, 89

Bertok, Peter, 23

Brecknell, Matthew, 13

Brown, Lawrie, 143

Brown, Ross, 43

Counsell, Steve, 3

Darcy, Peter, 133

Dekker, Anthony, 127

Downing, Nicholas, 61

Fani Marvasti, Amin, 109

Goldman, Grigori, 143

Karunasekera, Shanika, 89

Kelly, Paul, 79

Koehler, Henning, 71

Kulkarni, Santosh, 89

Lee, Lung-Chen, 163

Li, Ling, 53

Lutteroth, Christof, 163

Maire, Frederic, 43

Mans, Bernard, iii

Moffat, Alistair, 117

Mukherjee, Anshuman, 23

Nantes, Alfredo, 43

Nguyen, Duc Hoai, 153

Noble, James, 3

Pearce, David, 79

Pohl, Stefan, 117

Pupunwiwat, Prapassara, 99

Reynolds, Mark, iii

Rianto, Sugeng, 53

Roe, Paul, 13

Sattar, Abdul, 133

Skillicorn, David, 109

Stantic, Bela, 99, 133

Strooper, Paul, 153

Stuckey, Peter J., 61

Suess, Jorn Guy, 153

Tari, Zahir, 23

Tempero, Ewan, 3

Weber, Gerald, 163

Wirth, Anthony, 61

Wuensche, Burkhard, 33

Xie, Xin, 33

Zobel, Justin, 117

## 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 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 85 - Advances in Ontologies 2007**

Edited by Thomas Meyer, Meraka Institute, South Africa and Abhaya Nayak, Macquarie University, Australia. December, 2007. 978-1-920682-66-8.

Contains the proceedings of the 3rd Australasian Ontology Workshop (AOW 2007), Gold Coast, Queensland, Australia.

**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 2007, Adelaide, Australia.

**Volume 87 - Data Mining and Analytics 2008**

Edited by John F. Roddick, Jiuyong Li, Peter Christen and Paul Kennedy. November, 2008. 978-1-920682-68-2.

Contains the proceedings of the 7th Australasian Data Mining Conference (AusDM 2008), Adelaide, Australia. December 2008.

**Volume 88 - Koli Calling 2007**

Edited by Raymond Lister University of Technology, Sydney and Simon University of Newcastle. November, 2007. 978-1-920682-69-9.

Contains the proceedings of the 7th Baltic Sea Conference on Computing Education Research.

**Volume 89 - Australian Video**

Edited by Heng Tao Shen and Michael Frater. October, 2008. 978-1-920682-70-5.

Contains the proceedings of the 1st Australian Video Conference.

**Volume 90 - Advances in Ontologies**

Edited by Thomas Meyer, Meraka Institute, South Africa and Mehmet Orgun, Macquarie University, Australia. September, 2008. 978-1-920682-71-2.

Contains the proceedings of the Knowledge Representation Ontology Workshop (KROW 2008), Sydney, September 2008.

**Volume 91 - Computer Science 2009**

Edited by Bernard Mans Macquarie University. January, 2009. 978-1-920682-72-9.

Contains the proceedings of the Thirty-Second Australasian Computer Science Conference (ACSC2009), Wellington, New Zealand, January 2009.

**Volume 92 - Database Technologies 2009**

Edited by Xuemin Lin, University of New South Wales and Athman Bouguettaya, CSIRO. January, 2009. 978-1-920682-73-6.

Contains the proceedings of the Twentieth Australasian Database Conference (ADC2009), Wellington, New Zealand, January 2009.

**Volume 93 - User Interfaces 2009**

Edited by Paul Calder Flinders University and Gerald Weber University of Auckland. January, 2009. 978-1-920682-74-3.

Contains the proceedings of the Tenth Australasian User Interface Conference (AUIC2009), Wellington, New Zealand, January 2009.

**Volume 94 - Theory of Computing 2009**

Edited by Prabhhu Manyem, University of Ballarat and Rod Downey, Victoria University of Wellington. January, 2009. 978-1-920682-75-0.

Contains the proceedings of the Fifteenth Computing: The Australasian Theory Symposium (CATS2009), Wellington, New Zealand, January 2009.

**Volume 95 - Computing Education 2009**

Edited by Margaret Hamilton, RMIT University and Tony Clear, Auckland University of Technology. January, 2009. 978-1-920682-76-7.

Contains the proceedings of the Eleventh Australasian Computing Education Conference (ACE2009), Wellington, New Zealand, January 2009.

**Volume 96 - Conceptual Modelling 2009**

Edited by Markus Kirchberg, Institute for Infocomm Research, A\*STAR, Singapore and Sebastian Link, Victoria University of Wellington, New Zealand. January, 2009. 978-1-920682-77-4.

Contains the proceedings of the Fifth Asia-Pacific Conference on Conceptual Modelling (APCCM2008), Wollongong, NSW, Australia, January 2008.

**Volume 97 - Health Data and Knowledge Management 2009**

Edited by James R. Warren, University of Auckland. January, 2009. 978-1-920682-78-1.

Contains the proceedings of the Third Australasian Workshop on Health Data and Knowledge Management (HDKM 2009), Wellington, New Zealand, January 2009.

**Volume 98 - Information Security 2009**

Edited by Ljiljana Brankovic, University of Newcastle and Willy Susilo, University of Wollongong. January, 2009. 978-1-920682-79-8.

Contains the proceedings of the Australasian Information Security Conference (AISC 2009), Wellington, New Zealand, January 2009.

**Volume 99 - Grid Computing and e-Research 2009**

Edited by Paul Roe and Wayne Kelly, QUT. January, 2009. 978-1-920682-80-4.

Contains the proceedings of the Australasian Workshop on Grid Computing and e-Research (AusGrid 2009), Wellington, New Zealand, January 2009.

**Volume 100 - Safety Critical Systems and Software 2007**

Edited by Tony Cant, Defence Science and Technology Organisation, Australia. December, 2008. 978-1-920682-81-1.

Contains the proceedings of the 13th Australian Conference on Safety Critical Systems and Software, Canberra Australia.

**Volume 101 - Data Mining and Analytics 2009**

Edited by Paul J. Kennedy, University of Technology, Sydney, Kok-Leong Ong, Deakin University and Peter Christen, The Australian National University. November, 2009. 978-1-920682-82-8.

Contains the proceedings of the 8th Australasian Data Mining Conference (AusDM 2009), Melbourne Australia.