South African Computer Journal

Number 24
November 1999

Suid-Afrikaanse Rekenaar-Tydskrif

Nommer 24
November 1999

Special Issue: SAICSIT '99

The South African Computer Journal
An official publication of the Computer Society of South Africa and the South African Institute of Computer Scientists

Die Suid-Afrikaanse Rekenaartydskrif
'n Amptelike publikasie van die Rekenaarvereniging van Suid-Afrika en die Suid-Afrikaanse Instituut vir Rekenaarwetenskaplikes

World-Wide Web: `http://www.cs.up.ac.za/sacj/`

Editor
Prof. Derrick G. Kourie
Department of Computer Science
University of Pretoria, Hatfield 0083
dkourie@cs.up.ac.za

Sub-editor: Information Systems
Prof. Niek du Plooy
Department of Informatics
University of Pretoria, Hatfield 0083
nduplooy@econ.up.ac.za

Production Editors
Andries Engelbrecht
Department of Computer Science
University of Pretoria, Hatfield 0083

Herna Viktor
Department of Informatics
University of Pretoria, Hatfield 0083
sacj_production@cs.up.ac.za

Editorial Board

Prof. Judith M. Bishop
University of Pretoria, South Africa
jbishop@cs.up.ac.za

Prof. Richard J. Boland
Case Western University, U.S.A.
boland@spider.cwrv.edu

Prof. Trevor D. Crossman
University of Natal, South Africa
crossman@bis.und.ac.za

Prof. Donald. D. Cowan
University of Waterloo, Canada
dcowan@csg.uwaterloo.ca

Prof. Jürg Gutknecht
ETH, Zürich, Switzerland
gutknecht@inf.eth.ch

Prof. R. Nigel Horspool
University of Victoria, Canada
nigelh@csr.csc.uvic.ca

Prof. Fred H. Lochovsky
University of Science and Technology, Hong Kong
fred@cs.ust.hk

Prof. Kalle Lyytinen
University of Jyvaskya, Finland
kalle@cs.jyu.fi

Dr. Jonathan Miller
University of Cape Town, South Africa
jmiller@gsb2.uct.ac.za

Prof. Mary L. Soffa
University of Pittsburgh, U.S.A.
soffa@cs.pitt.edu

Prof. Basie H. von Solms
Rand Afrikaanse Universiteit, South Africa
basie@rkw.rau.ac.za

Subscriptions

	Annual	Single copy
Southern Africa	R80.00	R40.00
Elsewhere	US$40.00	US$20.00

An additional US$15 per year is charged for airmail outside Southern Afr

South African Computer Journal

Number 24, November 1999
ISSN 1015-7999

Suid-Afrikaanse Rekenaar-tydskrif

Nommer 24, November 1999
ISSN 1015-7999

Contents

Preface
P. Machanick 1

Research Articles

Active Learning: Issues and Challenges for Information Systems and Technology
RD Quilling, GJ Erwin and O Petkova 5

A Generic Modelling Framework for Interactive Authoring Support Environments
Paula Kotze 15

An Application of a Framework for Evaluation of the Factors Affecting Software Development Productivity in the Context of a Particular Organisational Environment
O Petkova and JD Roode 26

An Information-Theoretic Semantics for Belief Change
T Meyer 33

A Complexity Metrics Model for Software Correction
A Törn, T Andersson and K Enholm 40

A Conceptual Design for High-Volume Data Processing of Warehouse Database into Multidimensional Database
Paisarn Trakulsuk and Vichit Avatchanakorn 49

A Pragmatic Approach to Bitemporal Databases: Conceptualization, Representation and Visualisation
Chiyaba Njovu and WA Gray 58

A Building Recognition System
SP Levitt and B Dwolatzky 68

Computer Programming and Learning to Write
John Barrow 77

Co-operating to Learn using JAD Technologies
TA Thomas 87

Critical Success Factors for the Implementation of DSS at a Selection of Organisations in Kwazulu/Natal
URF Averweg and GJ Erwin 95

Enhancing the Predictability of Two Popular Software Reliability Growth Models
Peter A Keiller and Thomas A Muzzuchi 105

Generalised Unification of Finite Temporal Logic Formulas
Scott Hazelhurst 110

Harmonizing Global Internet Tax: A Collaborative Extranet Model
E Lawrence and B Garner 119

Improving Object Oriented Analysis by Explicit Change Analysis
Lui Yu, Siew Chee Kong, Yi Xun and Miao Yuan 128

Reconciling the Needs of New Information Systems Graduates and Their Employers in Small, Developed Countries
Rodney Turner and Glenn Lowry 136

Shortest Delay Scheduling Algorithm for Lossless Quality Transmission of Stored VBR Video under Limited Bandwidth
Fei Li, Yan Liu, Jack Yiu-Bun Lee and Ishfaq Ahmad 146

Software Croma Keying in an Immersive Virtual Environment
Frans van der Berg and Vali Lalioti 155

Some Automata-Theoretic Properties of $\bigcap$-NFA
Lynette van Zijl and Andries PJ van der Walt 163

Statistical Analysis of an Automated Computer Architecture Learning Environment
JT Waldron, J Horgan and G Keogh 168

The CILT Multi-Agent Learning System
Herna L Viktor 176

The Development of a Generic Framework for the Implementation of Cheap, Component-Based Virtual Video-Conferencing System
Soteri Panagou and Shaun Bangay 185

The Role of Experience in User Perceptions of Information Technology: An Empirical Examination
Meliha Handzic and Graham Low 194

What are Web Sites Used for: Cost Savings, Revenue Generating or Value Creating?
Man-Ying Lee 201

New Ideas Papers

Approaches to Video Transmission over GSM Networks
Bing Du and Anthony Maeder 210

From Information Security Baselines to Information Security Profiles
Rossouw von Solms and Helen van der Haar 215

Grounded Theory Methodology in IS Research: Glaser versus Strauss
J Smit 219

Introducing a Continuum of Abstraction-Led Hierarchical Search Techniques
Robert Zimmer and Robert Holte 223

Multimedia as a Positive Force to Leverage Web Marketing, with Particular Reference to the Commercial Sector
Stan Shear 229

Understanding HCI Methodologies
Peter Warren 234

Electronically Published Papers
http://www.cs.wits.ac.za/ ~philip/SAICSIT/SAICSIT-99/papers_ideas.html

---

Experience Papers

A Java Client/Server System for Accessing Arbitrary CANopen Fieldbus Devices via the Internet
Dieter Bühler, Gerd Nusser, Gerhard Gruhler and Wolfgang Küchlin 239

An Object-Oriented Framework for Rapid Client-side Integration of Information Management Systems
Ralf-Dieter Schimkat, Wolfgang Küchlin and Rainer Krautter 244

Distributed Operating Systems: A Study in Applicability
Jürgen Prange and Judith Bishop 249

Formal Verification with Natural Language Specifications: Guidelines, Experiments and Lessons so far
Alexander Holt 253

Introducing Research Methods to Computer Science Honours Students
Vashti Galpin, Scott Hazelhurst, Conrad Mueller and Ian Sanders 258

Visualising Eventuality Structure
ST Rock 264

Electronically Published Papers
http://www.cs.wits.ac.za/ ~philip/SAICSIT/SAICSIT-99/papers_exp.html

SAICSIT'99

South African Institute of Computer Scientists and Information Technologists

Annual Research Conference 17-19 November 1999

Prepare for the New Millennium

Is there life after y2k?

Mount Amanzi Lodge, Hartebeespoort, South Africa

Sponsors

PRICEWATERHOUSECOOPERS

Standard Bank

NRF

http://www.cs.wits.ac.za/

PHDS

research/PHDS.html

Think different.

Preface

Philip Machanick, Overall Chair: SAICSIT'99

Running SAICSIT'99, the annual research conference of the South African Institute for Computer Scientists and Information Technologists, has been quite an experience.

SAICSIT represents Computer Science and Information Systems academics and professionals, mainly those with an interest in research. When I took over as SAICSIT president at the end of 1998, the conference had not previously been run as an international event. I decided that South African academics had enough international contacts to put together an international programme committee, and a South African conference would be of interest to the rest of the world.

I felt that we could make this transition at relatively low cost, given that we could advertise via mailing lists, and encourage electronic submission of papers (to reduce costs of redistributing papers for review).

The first prediction turned out to be correct, and we were able to put together a strong programme committee.

As a result, we had an unprecedented flood of papers: 100 submitted from 21 countries. As papers started to come in, it became apparent that we needed more reviewers. It was then that the value of the combination of old-fashioned networking (people who know people) and new-fashioned networking (the Internet) became apparent. While the Internet made it possible to convert SAICSIT into an international event at relatively low cost, the unexpected number of papers made it essential to find many additional reviewers on short notice. Without the speed of e-mail to track people down and to distribute papers for review, the review process would have taken weeks longer, and it would have been much more difficult to track down as many new reviewers in so little time.

Even so, the number of referees who were willing to help on short notice was a pleasant surprise.

The accepted papers cover an interesting range of subjects, from management-interest Information Systems, to theoretical Computer Science, with subjects including database, Java, temporal logic and implications of e-commerce for tax.

In addition, we were very fortunate in being able invite the president of the ACM, Barbara Simons as a keynote speaker. Consequently, the programme for SAICSIT'99 should be very interesting to a wide range of participants.

We were only able to find place in the proceedings for 36 papers out of the 100 submitted, of which only 24 are full research papers. While this number of papers is in line with our expectation of how many papers would be accepted in each category, we did not have a hard cut-off on the number of papers, but accepted all papers which were good enough, based on the reviews. Final selection was made by myself as Programme Chair, and Derrick Kourie, as editor of the *South African Computer Journal*. Additional papers are published via the conference web site.

We believe that we have put together a quality programme, and hope you will agree.

Acknowledgments

I would like to thank the *South African Computer Journal* production team, Andries Engelbrecht and Herna Viktor, respectively from the Department of Computer Science and Informatics, University of Pretoria, for their work on producing the proceedings.

The reviewers listed overleaf did an excellent job: many wrote very detailed reports, sometimes after being called in on very short notice. Inevitably, there were some glitches resulting from the unexpected workload, but the buck stops with the programme chair: I promise to do better next time.

I would also like to thank my own department for putting up with the extra work and expense that running a conference entails. I tried not to burden them with too much extra work, but our secretaries, Zahn Gowar and Leanne Reddy, inevitably had to take on some extra work. John Ostrowick provided valuable assistance with design of our web pages and call for papers poster. Carol Kernick, who handles our finances and membership records, did a fine job of keeping up with the demands of the conference.

Finally, I would like to thank our sponsors, whose contribution made this conference been possible:

- PricewaterhouseCoopers – sponsored generous prizes and the conference banquet
- National Research Foundation (NRF) – provided financial support
- University of the Witwatersrand – provided financial support
- Programme for Highly Dependable Systems, University of the Witwatersrand – provided financial support
- Standard Bank – provided financial support

- Apple Computer – provided equipment for the conference
- Qualica – provided technical support including helping with the conference web site

Web Site

For more information about SAICSIT, including a pointer to the conference site, see <http://www.saicsit.org.za>.

Referees

- Department of Computer Science, University of Pretoria
  - Derrick Kourie
  - Bruce Watson
  - Vali Lalioti
  - Andries Engelbrecht
  - Ivan Mphahlele
- German National Research Center for Information Technology - GMD
  - Gernot Goebbels
- School of Math Stat and IT, Natal University Pietermaritzburg
  - Peter Warren
- Graduate School of Business, University of Cape Town
  - Kurt April
- CSIR
  - James Jardine
- Department of Computer Science, Rhodes University
  - Shaun Bangay
  - Peter Clayton
  - John Ebden
  - Richard Foss
  - George Wells
  - Peter Wentworth
- Information Technology Division, Rhodes University
  - Caro Watkins
- Department of Information Systems, Rhodes University
  - Brenda Mallinson
  - Dave Sewry
- Department of Informatics, University of Pretoria
  - Carina de Villiers
  - Herna Viktor
  - Niek du Plooy
  - Elsje van Rooyen
  - Machdel Matthee
  - Alan Abrahams
  - Jackie Phahlamohlaka
- School of Information Technology & Engineering, University of Ottawa
  - Stan Szpakowicz
  - Dwight Makaroff
- Department of Information Systems, Victoria University
  - Glenn Lowry
  - Peter Shackleton
  - Tas Adam
  - Alastair Wallace
  - Stephen Burgess
  - Dave Deal
  - Julie Fisher
  - Jerzy Lepa
  - Geoff Sandy
  - Rod Turner
  - Alastair Wallace
  - Andrew Wenn
- Peninsula School of Computing and Information Technology, Monash University
  - Ainslie Ellis
- School of Information Management and Systems, Monash University
  - Angela Carbone
- School of Computer Science & Software Engineering, Monash University
  - Dianne Hagan
  - Judy Sheard
- School of Management Information Systems, Deakin University
  - John Lamp
  - David Mackay
  - Philip Joyce

- Marie van der Klooster
- Bill Hewett
- Jennie Carroll
- Rodney Carr

- Computer and Automation Research Institute, Hungarian Academy of Sciences
  - Ferenc Vajda
- Department of Computer Science, University of the Witwatersrand
  - Ian Sanders
  - David Lubinsky
  - Conrad Mueller
  - Yinong Chen
  - Bob Baber
  - Vashti Galpin
  - András Salamon
  - Scott Hazelhurst
  - Philip Machanick
  - Zoltán Fazekas
- Department of Electrical Engineering, University of the Witwatersrand
  - Barry Dwolatzky
  - Farzin Aghdasi
- Department of Mathematics, University of the Witwatersrand
  - Jonathan Burgess
- School of Law, University of the Witwatersrand
  - Victoria Bronstein
- Edward Nathan & Friedland Attorneys, Johannesburg
  - Justine White
- Department of Information Systems & Technology, University of Durban-Westville
  - Geoff Erwin
- Department of Computer Science, Clemson University, Clemson, SC
  - Karen Hay
- Logical SA
  - Philip Green
- Department of Computer Science, University of Bristol
  - Alan Chalmers
- Department of Systems and Computer Science, Howard University, Washington, D.C
  - Todd Shurn
  - John Trimble
- Department of Computer Science, University of Cape Town
  - Sonia Berman
- King's College, London
  - Peter Wood
- Department of Information Systems, University of Cape Town
  - Paul Licker
  - Steve Erlank
  - Alemayehu Molla
- Department of Department of Mathematics and Applied Mathematics, University of Cape Town
  - Ingrid Rewitzky
  - Peter Jipsen
  - Renkuan Guo
- Department of Information Systems, University of the Western Cape
  - Andy Bytheway
- Department of Electrical Engineering and Computer Science, The George Washington University, Washington, DC
  - Rachelle Heller
- School of Biophysical Sciences and Electrical Engineering, Swinburne University of Technology
  - Ian Macdonald
- Electrical and Electronic Systems Engineering, Queensland University of Technology
  - Anthony Maeder
- Computer Science Department, Åbo Akademi
  - Aimo Törn
  - Kaisa Sere
  - Mats Aspnas
- Division of Informatics, University of Edinburgh
  - Graeme Ritchie
  - Arturo Espinosa-Romero
  - Jon Oberlander

- Chris Brew
- Alexander Holt

- Department of Computer Science and Engineering, Helsinki University of Technology
  - Tommi Junttila
  - Nisse Husberg
- Technical University of Budapest, Department of Measurement and Information Systems
  - Andras Pataricza
- NASA-Goddard Space Flight Center
  - Nigel Ziyad
- Department of Computer Science & Information Systems, University of Port Elizabeth
  - Janet Wesson
  - Leon Nicholls
- Computer Technology Department, Indiana University Purdue University Indianapolis
  - Tim Price
- Department of Electrical Engineering and Computer Science, The University of Michigan
  - Trevor Mudge
- Department of Electrical Engineering and Computer Sciences, University of California, Berkeley
  - David Forsyth
- School of Computer Science, University of Birmingham
  - Mark Ryan
- Faculty of Mathematics, Computer Science, Physics & Astronomy, University of Amsterdam
  - Carlos Areces
- Department of Computer Science, Universidade do Vale do Rio dos Sinos - UNISINOS Rio Grande do Sul
  - Marcelo Walter
- School of Information Technology & Mathematical Sciences, University of Ballarat
  - Binh Pham

Introducing a Continuum of Abstraction-Led Hierarchical Search Techniques

Robert Zimmer[a] and Robert Holte[b]

[a]Brunel University, Uxbridge, Middx, UB8 3PH, England, Robert.Zimmer@brunel.ac.uk
[b]University of Ottawa, Ottawa, Ontario, Canada, holte@site.uottawa.edu

Abstract

Abstraction in search works by replacing a state space by another space (the abstract space) that is easier to search, either because it is smaller or because it contains more solutions. Results of searches in the abstract space are then used to guide searches in the original space in one of two distinct ways. The first uses the lengths of the abstract solutions as a heuristic for an A search of the original space: this always produces optimal solutions. The second uses the steps in the abstract solution as sub-goals for the search in the original space: this strategy does not guarantee optimality, but it does tend to find a solution quickly. In this paper, both of these types of abstraction-led search are explained, and a general setting for all abstraction-led search is introduced. The two standard cases are included as extreme cases in the general setting, and so are a continuum of new techniques between the two. It is hoped that further study of this continuum will lead to new, useful heuristic search techniques.*

Keywords: *Heuristic Search, Abstraction*
Computing Review Categories: *I.2.8, G.2.2*

1 Introduction

Heuristic search is ubiquitous in AI. A particular form of heuristic search, state-space search, is the cornerstone of many AI systems, including most planning and problem solving systems. Consequently, techniques for speeding up heuristic search, or for automatically generating or improving heuristics, are of central importance to AI. Abstraction is a widely studied means of speeding up state-space search. Instead of directly solving a problem in the original search space, the problem is mapped into and solved in an abstract search space. The abstract solution is then used to guide the search for a solution in the original space.

This paper draws together two separate strands of research in abstraction-led heuristic search. The two strands diverge when deciding how the results of the search in the abstract space are used to guide the search in the original space. One method of hierarchical search uses the length of the solution in the abstract space as a heuristic for A^* [7]. See, for example, Holte et al. [9], Gaschnig [5], Pearl [14] and Guida and Somalvico [6]. The attraction of this method is that most commonly-used techniques of abstraction automatically produce admissible heuristics [16]. Therefore, this method of hierarchical search is guaranteed to produce optimal solutions.

The other widely-studied method of hierarchical search uses the individual steps in the abstract solution (the solution in the abstract space) as a sequence of subgoals to be solved. The solutions of these subgoals are then linked together to form the final solution [10, 13, 17, 19, 12]. The abstract solution serves as a skeleton for the final solution; the process of fleshing out the skeleton to a complete solution is called *refinement*. This method has the attraction of being very fast; it has the disadvantage that the solutions it produces are not guaranteed to be optimal.

In this paper we introduce a framework for seeing these two techniques as variants of the same algorithm: the techniques simply differ in the values of two numerical parameters. By introducing this framework, we open up a whole world of search techniques: each valuation of the parameters defines another search algorithm. We shall see, for example, that two other well-studied heuristic search techniques also appear within the setting.

Before discussing the use of abstractions in search, we turn to a discussion of what a graph abstraction is.

2 Abstractions of Graphs

There are two different axes along which graphs can be abstracted. In this chapter, these two types of abstraction are defined with reference to an example that makes clear the Artificial Intelligence genesis of this research. The graph, which follows as Figure 1, represents a two disk tower of Hanoi[1] problem. The two disks are called S and L (for small and large), and the states are named by the list of disks on each peg. An example of such a state is <S,L,_ >,

[1]In the Towers of Hanoi puzzle there are three pegs and N different sized disks (here N is 2) sitting on the pegs so that no disk is sitting on a smaller one. The top disk on a peg may be moved onto an empty peg or onto the top of any peg whose top disk is smaller than the one being moved. the goal is to move all the disks from the first peg to the third peg, without violating the size-order.

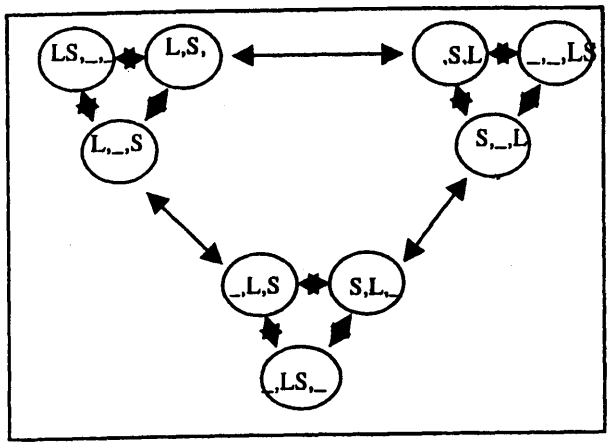

Figure 1:

which is the position in which the small disk is on the first peg and the large disk is on the second peg.

The states of the graph do not give the complete representation of the system: we also need to name the arrows. The choice of names for the arrows can make an enormous difference to the effectiveness of abstraction [20]. In the example in this paper, we will assume that there are four operators, that is four distinct names of arrows. These names are: move small clockwise, move small anti-clockwise, move large clockwise, move large anti-clockwise. It should be noted that the first two operators are defined in every state, while the second two are defined but rarely. The state graph is given below.

The graph is laid out in a way that highlights the recursive nature of the problem: the three small triangles in each corner represent one-disk problem. The graph of a three disk problem is a larger triangle with the graph in Figure 1 in each corner. Seeing the graph in this form emphasises the structure of the problem, and makes it easy to see a solution. However, the graph is unlikely to be presented, either to a person or to a computer, in so clearly regular a fashion. One of the forms of abstraction homomorphic abstraction works (when it works best) by discovering such regularities.

2.1 Homorphic Abstraction

The simple idea of homorphic abstraction is that the abstract space is derived by joining several states in the original space into one abstract state. This will clearly make the new space smaller, and therefore easier to search. The condition that needs to be satisfied to make this process work is given below:

> If two states, s and s', get put into the same abstract class, and an operator Op is defined on both states, then the state arrived at by applying the operator in s must be the same as the state reached by applying the operator in s'. That is Op(s) and Op(s') must be put into the same class.

This condition is necessary for the operator to be unambiguously defined in the abstract state: Op applied to the abstract space containing s is defined to be the abstract state containing Op(s).

This notion of abstraction gets its name from the observation that the above property is equivalent to saying that there is a morphism from the original space to the abstract space. A *homomorphism* is a mapping, f, of the vertices and arrows of the graph such that: if e:a → b then f(e):f(a) → f(b). The algebraically inclined reader will be interested to note that this is related to notions of covering and division in the algebraic theories of Automata and Concrete Categories [2, 20].

To see how these abstractions work in practice, we turn again to the example. Suppose that the states <LS,_,_> and <L,S,_> were joined to make an abstract state. Now, since moving S clockwise is defined in both, the results <L,S,_> and <L,_,S> need to be in the same abstract class as each other. In this case, we find that the abstract class needs to contain all of <LS,_,_>, <L,S,_>, and <L,_,S>. Put another way, the entire upper-left hand corner collapses to a single state. There are no other consequences of that initial decision. Therefore, one of the abstractions of the space, has one state in the upper-left hand corner and still the triangles in the others. Suppose on the other hand, we had decided to join <LS,_,_> and <_, LS,_>. The consequences of this decisiondictated by the operators that move the small diskare that <L,S,_> and <_, L,S> need to be joined, and that <L,_,S> and <S, L,_> need to be joined. This is the all abstraction that needs to be done. Therefore, another abstraction of the original space collapses the two top triangles into a single triangle, and leaves the bottom triangle as it is.

We have developed a system that finds good homomorphic abstractions automatically and uses them to solve graph searching problems (see [10] for details). The technique we use in the system is called *max-degree star abstraction*.

2.1.1 Star Abstraction

Max degree star abstraction is very straight forward: the state with the largest degree (that is, the states which is connected to the most other states) is grouped together with its neighbours within a certain distance (the "abstraction radius") to form a single abstract state. This is repeated until all states have been assigned to some abstract state. Having thus created one level of abstraction the process can be repeated recursively until a level is created containing just one state. This forms an abstraction hierarchy whose top-level is the trivial search space. The bottom level of the hierarchy is the original search space. We will see later how these hierarchies lead to searches. Before that, we turn to the other kind of abstraction.

2.2 Embedding Abstractions

In most AI search systems, a search space is not defined explicitly as a graph. Instead, the graph is defined implicitly, typically in the STRIPS notation [3]. In the STRIPS notation a state is defined as the set of sentences true in itthe sentences are given in a formal language (containing constants, variables, predicate symbols, etc.). The successor relation between states is represented by operators that map one state to another by adding to or deleting from the set of sentences (i.e. the state) to which it is applied. Each operator has preconditions, stated in the formal language, specifying to which states the operator may be applied.

The natural notion of abstraction giving this formulation is to remove symbols from the formal language or remove pre-conditions from the definitions of the operators (e.g. see [12]). From the point of view of the graph, the effect of removing preconditions on operators is to allow more arrows. Thus in the tower of hanoi, for example, you can abstract the space by allowing the large disk to move even if it has the small disk on top of it. In this kind of abstraction the search for solutions is eased by the greater number of solutions available.

In the next section we turn to heuristic search, and the part that this easier abstract search can play in it.

3 A^* and Hierarchical A^*

One of the ways that abstract searches guide concrete ones is by providing heuristics for a standard AI search technique called A^*. A^* is a variant of heuristic best-first search that is guaranteed to produce optimal solutions, under a condition on the heuristic.

3.1 Heuristic Best First Search

To understand the rest of the paper it is important to be quite precise about what, among the possibilities, we mean by a best first search. In this section, as in the whole paper, the problem is to find a goal state: either a particular state or a state satisfying a particular property. To be able to employ a best-first heuristic search, we need an estimate (or heuristic) of how far each state is from a goal state. The estimate of the distance from a state, S, to a goal state is denoted h(S). At any time we keep track of all the states we've been to so far. Always, the next state to be visited is the most promising of all the successors of all the states weve visited. This will give us our best chance of finding a goal state soon. A^* is exactly this kind of search, but with the heuristic slightly changed to take account of a different objective. In A^* we are not interested in finding a goal soon, but in finding a shortest path to a goal. This alters the heuristic we use.

3.2 A^*

In A* the estimate of the value of a state, s, is the sum of the distance from the start state to s, g(s), and the heuristic

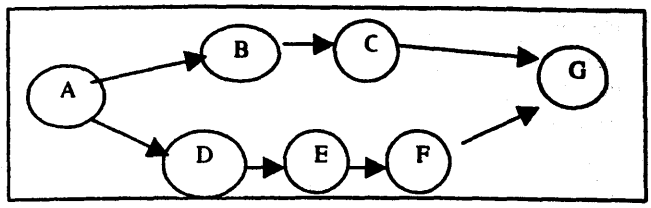

Figure 2:

estimating the distance to a goal state. This seems a reasonable expression to use for a heuristic since we are trying to minimise the length of the path, which we are estimating as g(s) + h(s). The effectiveness of A^* is bound up with the notion of an admissible heuristic.

Definition: A heuristic is said to be admissible if its value is never greater than the true value.

That is, h is admissible if it is consistently an underestimator. The main result is that if h is an admissible heuristic, then A^* will find an optimal path. To see why this is the case, consider the following graph segment:

Assume that it took 5 moves to get to A, B is correctly estimated to be 2 moves away from G, and D is underestimated to be 1 move away from G. Then Bs value is 7 and Ds is 6, so D is visited next. Then, if h(E) is greater than 1, value(A) will be greater than 7 and B will be visited next. The most confused that things could be here is if h(E) were 1 as well. Then value(E) = value(B). So F *may* be visited next. However, after that, F *must* be worse than B to expand. And so we finally get to B and the best path.

The property that makes this argument work is exactly the admissibility of h. If h were not assumed admissible, then B may have been estimated to be too far away from the goal ever to be visited.

3.3 Abstraction and A^*

The way that abstraction leads to heuristics is as follows: at the concrete level define h(s) to be the actual shortest distance (in the abstract space) from the class that contains s to the nearest class that contains a goal. This heuristic will always be admissible. This is true because in embedded abstractions all of the real paths in the concrete space are available in the abstract space, and in homomorphic abstractions all of the paths in the concrete space are available in the abstract space (either in the same or a shortened form). Therefore, the shortest path in the abstract space will never be longer than the shortest path in the concrete space. It, therefore, seems very inviting to use real values in the abstract space as heuristic values in the concrete search. Unfortunately, this doesn't always work as well as one might hope.

Suppose that we are doing an abstraction-led A^* search. And suppose further that the values in the abstract space are computed by blind search. Then, if the abstraction is an embedding abstraction, it turns out that the A^* search will need to open all of the nodes that would have been opened in a blind search of the concrete space [18].

However it has been shown that abstract-led A^* can speed up search, if the abstraction is a homomorphic one [9]. We, therefore, confine ourselves to homomorphic abstractions for the rest of the paper.

There is another way to use homomorphic abstractions to guide searching in the concrete space, and that is called refinement.

4 Refinement

The idea of refinement is to turn the original search in the concrete space into a series of small searches. The abstract space gives you a skeleton of a solution. To flesh out the skeleton you need to search within each class to find the right state to lead onto the next skeletal path. Consider, for example, the tower of hanoi graph again. If you wanted to get from <LS,_,_> to <_,_,LS>. At the highest useful level of abstraction, the graph is simply a triangle, in which each of the corner triangles has collapsed to a single state. The search at that level is very easy: and not much searching shows that the right move is along the top. Now to flesh this out. Starting from <LS,_,_>, to be able to go across the top, you first need to get to the state <L,S,_>. This leads to another small search. The way this will work in practice is that a large tower of abstractions will be built, and each abstraction will provide a skeleton for searches in the state space below.

In [9] it was pointed out that refinement and A^* have several points in common. It is the contribution of this paper to make to turn these commonalties into continuous spectra.

5 Unifying Hierarchical A^* and Refinement

The main connection between refinement and A^* is that both are guided by h(S), the abstract distance (on the solution path found) from f(S) to f(Goal).

The first difference between refinement and A^* is how h(S) is used in computing a states priority. In A^* h(S) is added to g(S), the distance from the start state to S, to compute priority, whereas in refinement h(S) is used by itself.

The second difference between refinement and A^* is in what they do when they encounter a state S for which h(S) is not known. Refinement does just one search at the abstract level. This generates h values for a certain set of states and refinement's search is confined to this set. Refinement ignores all states outside this set (i.e. all states whose h value is not determined by the first abstract search). A^* is the exact opposite. Every time it encounters a state S for which h(S) is not known it initiates a search at the abstract level in order to determine h(S).

These are the only two differences between A^* and refinement. Although they might at first seem to be qualitative differences, each can be formulated quantitatively, i.e. in terms of a numerical parameter whose value can be varied continuously between a value corresponding to A^* and a value corresponding to refinement. The parameter associated with the first difference will be called W, the parameter associated with second difference P.

5.1 Search Parameters

Search parameter W is a familiar one in A^* research. In A^*, the "priority" of a state, f(S), combines two distance measures: g(S) and h(S). In normal A^*, these two factors are given equal weight: value(N) = g(N) + h(N). Various researchers have explored the effects of weighting these two factors differently [15, 4]. In general, then, value(N) = W*g(N) + (1-W)*h(N), where W is a parameter the user can set. Normal A^*, which weighs g and h equally, corresponds to W=0.5.

To get refinement , set W=0. Then f(S) is based entirely on h(S). This will correspond to refinement providing that states whose h values are equal are searched in a breadth first manner.

The second parameter characterising the continuum of search techniques is a probability, P. To understand what P means, consider what happens when search at some level reaches a state S for h(S) is not known. In this circumstance A^* **always** initiates a new search in the abstract space in order to compute h(S). By contrast refinement **never** does so: if h(S) is not determined by the very first search at the abstract level, then S is ignored. In between these two extremes are search techniques that **sometimes** initiate a new search and sometimes do not. The parameter P specifies the probability that a new search will be initiated at the abstract level when a state with an unknown h value is encountered . Thus, for A^*, P = 1.0 and for refinement P = 0.0.

The definition of the P parameter as the **probability** of opening or rejecting a state during search, is intended simply as a first "rough cut". In practice, the choice of nodes to open would likely be better made on a reasoned basis than by chance. The significance of the P parameter lies not in the specific definition we have given but in introducing the concept that a search system may choose (somehow) to add to the OPEN list only some of the states that A^* would add.

These two parameters have been added to the hierarchical A^* system called "V3" in [9] This system itself is an ordinary A^* with a few special caching techniques added to reduce duplicate effort during hierarchical search. The W parameter is obviously introduced into the computation of value. The P parameter is introduced into the step in A* where states are added to the OPEN list, since it is at this step that the h(-) value is required. States whose h value is known are added as usual. For each state S at this step for which h(S) is not known a probabilistic decision is made: S is ignored with probability 1-P and, with probability P, h(S) is computed by searching at the abstract level and then S is added to OPEN. The only exception to this rule is the start state: h(Start) is always computed so that Start can be added to OPEN to initiate a search. The computation

of h(Start) is what causes the first search to be done at the abstract level; for refinement, or any other technique for which P=0, this is the only search at the abstract level.

6 Some Principal Variations

Certain combinations of W and P correspond to familiar systems. P = 1.0 and W = 0.5 is ordinary A^*, of course. P = 1.0 and W = 1.0 is blind search (A^* with h(S)=0 for all S). But it is a particularly inefficient implementation since it actually computes h(S) for every S encountered during search and then multiplies the h(S) value by 0 when computing f(S). The Graph Traverser [1] corresponds to P = 1.0 and W = 0.

When P = 0.0 one search is made at each abstract level from start to goal. All subsequent search is at the base level and is confined to the portion of the state space that corresponds to the abstract solution path. Different values for W correspond to different strategies for searching within this portion of state space. W=0 is ordinary refinement. W= 1.0 conducts this search in a breadth first manner without regard for a state's h value. This method is called "optimal refinement" (OptR) in [10] because it finds the shortest possible refinement of the given abstract path.

Another method that finds an optimal refinement corresponds to W = 0.5. This method will be called "refinement by A^*" (Ref-A^*) because it conducts the search like A^*, giving equal weight to g(-) and h(-). It differs from A^* in that its search is confined the portion of state space corresponding to the first (and only) abstract solution path. Note that if the shortest refinement is not unique, OptR and Ref-A^* might produce different refinements, say Ref1 and Ref2. At the next level down in the abstraction hierarchy, Ref1 will be used by OptR to constrain search, but Ref2 will be used by Ref-A^*. At this level, the shortest refinement of Ref1 might be a different length than the shortest refinement of Ref2. Thus, in an abstraction hierarchy with severals OptR and Ref-A^* do not necessarily produce optimal solutions, or even equal length solutions. In preliminary experiments, we found that the two techniques do produce solutions of very similar lengths and that Ref-A^* usually does less work.

7 Conclusions and Future Work

This paper has surveyed the use of abstractions in determining heuristic searches of directed graphs. It described two kinds of abstraction: embedding abstraction and homomorphic abstraction. Results indicate that homomorphic abstraction is the more useful of the two for searching. The paper has also presented two kinds of abstraction-led search: Hierarchic A^* and Refinement. These are both well-studied. The first always produces optimal solutions and the second tends to produce good solutions quickly. The main contribution of this paper is to introduce two continuous parameters that characterise the differences between these two methods.

Because the parameters are continuous, it follows that there is a continuum of search techniques that are between the two well-known ones. By moving continually through the space of search techniques, from Hierarchic A^* to Refinement, we would be moving smoothly between optimal solution producing techniques to fast working techniques. It should, in future, be possible to set your search techniques to match your needs for speed versus accuracy.

Much empirical and theoretical exploration of the continuum of search techniques will be done in the future. The theoretical work will help us understand the dynamics of search. The empirical work may well lead to new search techniques that are almost as fast as refinement and produce results that are almost as good as Hierarchic A^*.

References

[1] JE Doran and D Michie, *Experiments with the Graph Traverser Program*, Proceedings of the Royal Society of London, Series A, **294**:235-259, (1966).

[2] Eilenberg, Automata, languages and machines, Vol. B, Academic Press, (1976).

[3] RE Fikes and NJ Nilson, *STRIPS: a New Approach to the Application of Theorem Proving to Problem Solving*, Artificial Intelligence, **2**(3-4):189-208, (1971).

[4] J Gaschnig, *Exactly How Good are Heuristics?: Toward a Realistic Predictive Theory of Best-first Search*, Proc. IJCAI'77, pp434-441, (1977).

[5] J Gaschnig, *A Problem Similarity Approach to Devising Heuristics: First Results*, Proc. IJCAI'79, pp301-307, (1979).

[6] G Guida and M Somalvico, *A method for computing Heuristics in Problem Solving*, Information Sciences, **19**:251-259, (1979).

[7] PE Hart, NJ Nilsson and B Raphael, *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*, IEEE Transactions on Systems Science and Cybernetics, **4**(2), pp100-107, (1968).

[8] RC Holte, C Drummond, MB Perez, RM Zimmer and AJ MacDonald, *Searching with Abstractions: A Unifying Framework and New High- Performance Algorithm*, Proc. of the 10th Canadian Conference on Artificial Intelligence (AI'94), Morgan Kaufman Publishers, pp263-270, (1994).

[9] RC Holte, MB Perez, RM Zimmer and AJ MacDonald, *Hierarchical A*: Breaking Valtortas Barrier*, Proceedings AAAI'96, (19996).

[10] RC Holte, T Mkadmi, RM Zimmer AJ MacDonald, *Speeding Up Problem-Solving by Abstraction: A Graph-Oriented Approach*, Artificial Intelligence, **85**:321-361, (1996).

[11] L Kavraki and J-C Latombe, *Randomized Preprocessing of Configuration Space for Fast Path Planning*, Proceedings of the IEEE International Conference on Robotics and Automation, (1994).

[12] CA Knoblock, *Automatically Generating Abstractions for Planning*, Artificial Intelligence, **68**(2):243-302, (1994).

[13] M Minsky, *Steps Toward Artificial Intelligence*, in Computers and Thought, E Feigenbaum and J Feldman (eds.), McGraw-Hill, pp406-452, (1963).

[14] J Pearl, *Heuristics*, Addison-Wesley, (1984).

[15] I Pohl, *Heuristic Search Viewed as Path Finding in a Graph*, Artificial Intelligence, **1**(3):193-204, (1970).

[16] A Prieditis, *Machine Discovery of Admissible Heuristics*, Machine Learning, **12**:117-142, (1993).

[17] E Sacerdoti, *Planning in a hierarchy of abstraction spaces*, Artificial Intelligence, **5**(2):115-135, (1974).

[18] M Valtorta, *A result on the computational complexity of heuristic estimates for the A* algorithm*, Information Sciences, **34**:48-59, (1984).

[19] Q Yang and JD Tenenberg, *Abtweak: Abstracting a nonlinear, least commitment planner*, Proc. AAAI'90, pp204-209, (1990).

[20] Robert Zimmer, *Decomposing Concrete Categories*, to appear in Formal Aspects of Computing, (1999).

South African Computer Journal

Number 24, November 1999
ISSN 1015-7999

Suid-Afrikaanse Rekenaartydskrif

Nommer 24, November 1999
ISSN 1015-7999

Notes for Contributors

The prime purpose of the journal is to publish original research papers in the fields of Computer Science and Information Systems, as well as shorter technical research notes. However, non-refereed review and exploratory articles of interest to the journal's readers will be considered for publication under sections marked as Communications of Viewpoints. While English is the preferred language of the journal, papers in Afrikaans will also be accepted. Typed manuscripts for review should be submitted in triplicate to the editor.

Form of Manuscript

Manuscripts for *review* should be prepared according to the following guidelines:

- Use wide margins and $1\frac{1}{2}$ or double spacing.
- The first page should include:
- the title (as brief as possible)
- the author's initials and surname
- the author's affiliation and address
- an abstract of less than 200 words
- an appropriate keyword list
- a list of relevant Computing Review Categories
- Tables and figures should be numbered and titled.
- References should be listed at the end of the text in alphabetic order of the (first) author's surname, and should be cited in the text according to the Harvard. References should also be according to the Harvard method.

Manuscripts accepted for publication should comply with guidelines as set out on the SACJ web page,

`http://www.cs.up.ac.za/sacj`

which gives a number of examples.

SACJ is produced using the LaTeX document preparation system, in particular LaTeX 2_ε. Previous versions were produced using a style file for a much older version of LaTeX, which is no longer supported. Please see the web site for further information on how to produce manuscripts which have been accepted for publication.

Authors of accepted publications will be required to sign a copyright transfer form.

Charges

Charges per final page will be levied on papers accepted for publication. They will be scaled to reflect typesetting, reproduction and other costs. Currently, the minimum rate is R30.00 per final page for contributions which require no further attention. The maximum is R120.00, prices inclusive of VAT.

These charges may be waived upon request of the author and the discretion of the editor.

Proofs

Proofs of accepted papers may be sent to the author to ensure that typesetting is correct, and not for addition of new material or major amendments to the text. Corrected proofs should be returned to the production editor within three days.

Letters and Communications

Letters to the editor are welcomed. They should be signed, and should be limited to about 500 words. Announcements and communications of interest to the readership will be considered for publication in a separate section of the journal. Communications may also reflect minor research contributions. However, such communications will not be refereed and will not be deemed as fully-fledged publications for state subsidy purposes.

Book Reviews

Contributions in this regard will be welcomed. Views and opinions expressed in such reviews should, however, be regarded as those of the reviewer alone.

Advertisement

Placement of advertisements at R1000.00 per full page per issue and R500.00 per half page per issue will be considered. These charges exclude specialised production costs, which will be borne by the advertiser. Enquiries should be directed to the editor.