Jean-Daniel Zucker
Lorenza Saitta (Eds.)

Abstraction, Reformulation and Approximation

LNAI 3607

6th International Symposium, SARA 2005
Airth Castle, Scotland, UK, July 2005
Proceedings

Springer

Lecture Notes in Artificial Intelligence 3607

Edited by J. G. Carbonell and J. Siekmann

Subseries of Lecture Notes in Computer Science

Jean-Daniel Zucker Lorenza Saitta (Eds.)

Abstraction, Reformulation and Approximation

6th International Symposium, SARA 2005
Airth Castle, Scotland, UK, July 26-29, 2005
Proceedings

Springer

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Volume Editors

Jean-Daniel Zucker
LIM&BIO, EPML-CNRS 32
Université Paris 13
74, rue Marcel Cachin, 93017 Bobigny, France
E-mail: zucker@limbio-paris13.org

Lorenza Saitta
Università del Piemonte Orientale
Dipartimento di Informatica
Via Bellini 25/G, 15100 Alessandria, Italy
E-mail: saitta@al.unipmn.it

Library of Congress Control Number: 2005929387

CR Subject Classification (1998): I.2, F.4.1, F.3

ISSN 0302-9743
ISBN-10 3-540-27872-9 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-27872-6 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11527862 06/3142 5 4 3 2 1 0

Preface

This volume contains the proceedings of the 6th Symposium on Abstraction, Reformulation and Approximation (SARA 2005). The symposium was held at Airth Castle, Scotland, UK, from July 26th to 29th, 2005, just prior to the IJCAI 2005 conference in Edinburgh. Previous SARA symposia took place at Jackson Hole in Wyoming, USA (1994), Ville d'Estrel in Qubec, Canada (1995), Asilomar in California, USA (1998), Horseshoe Bay, Texas, USA (2000), and Kananaskis, Alberta, Canada (2002). This was then the first time that the symposium was held in Europe. Continuing the tradition started with SARA 2000, the proceedings have been published in the LNAI series of Springer.

Abstractions, reformulations and approximations (AR&A) have found applications in a variety of disciplines and problems, including constraint satisfaction, design, diagnosis, machine learning, planning, qualitative reasoning, scheduling, resource allocation and theorem proving, but are also deeply rooted in philosophy and cognitive science. The papers in this volume capture a cross-section of the various facets of the field and of its applications. One of the primary uses of AR&A is oriented to overcome computational intractability. AR&A techniques, however, have also proved useful for knowledge acquisition, explanation and other applications, as papers in this volume also illustrate.

The aim of SARA is to provide a forum for intensive and friendly interaction among researchers in all areas of AI in which an interest in the different aspects of AR&A exist. The diverse backgrounds of participants at this and previous meetings led to rich and lively exchanges of ideas, allowed the comparisons of goals, techniques and paradigms, and helped identify important research issues and engineering hurdles. SARA has always invited distinguished members of the research community to present keynote talks. SARA 2005 was no exception to this rule with invited talks from Rada Chirkova of the North Carolina State University at Raleigh, USA Aristide Mingozzi of the University of Bologna, Italy, and Robert Zimmer of Goldsmiths College, University of London and Goldsmiths Digital Studios, London.

We would like to thank the authors of all the submitted papers and research summaries, the referees, the invited speakers, the Program Committee members for all their time and effort, and, of course, all the attendees. We also thank the members of the Steering Committee for their advice along the way. In addition, a great "merci" to the Local Chair Ian Miguel and to all those who contributed to the organization of SARA 2005, in particular Mélanie Courtine.

Paris, May 19, 2005

Jean Daniel Zucker
Lorenza Saitta

Organization

http://sara2005.limbio-paris13.org

SARA 2005 was organized by Ian Miguel.

Executive Committee

Conference Chair	Jean-Daniel Zucker, University of Paris 13
	Lorenza Saitta, Universitá del Piemonte Orientale
Organizing Chair	Ian Miguel, University of York
Proceeding Chair	Mélanie Courtine, University of Paris 13

Program Committee

J. Christopher Beck, University of Toronto
Berthe Y. Choueiry, University of Nebraska-Lincoln
Stefan Edelkamp, Albert Ludwigs University Freiburg
Tom Ellman, Vassar College
Jérôme Euzenat, INRIA Rhône-Alpes
Mike Genesereth, Stanford University
Robert C. Holte, University of Alberta
Daniel Kayser, University of Paris Nord
Sven Koenig, University of Southern California
Michael Lowry, NASA Ames Research Center
Hiroshi Motoda, Osaka University
Pandurang Nayak, PurpleYogi.com
Doina Precup, McGill University
Peter Revesz, University of Nebraska-Lincoln
Marie-Christine.Rousset, University of Paris XI
Bart Selman, Cornell University
Barbara Smith, University College Cork
Miroslav Velev, CMU
Toby Walsh, Cork Constraint Computation Centre, University College Cork
Robert Zimmer, Goldsmiths College, University of London
Weixiong Zhang, Washington University in St Louis

Steering Committee

Berthe Y. Choueiry, University of Nebraska-Lincoln
Tom Ellman, Vassar College

Mike Genesereth, Stanford University
Fausto Giunchiglia, University of Trento and ITC-IRST
Alon Halevy, University of Washington
Robert Holte, University of Alberta
Sven Koenig, Georgia Institute of Technology
Michael Lowry, NASA Ames Research Center
Pandurang Nayak, PurpleYogi.com
Jeffrey Van Baalen, University of Wyoming
Toby Walsh, Cork Constraint Computation Centre, University College Cork

Supplementary Referees

Yann Chevaleyre
Olivier Cogis
James Ezick
Attilio Giordana
Joel Gompert
Brahim Hnich
Shahid Jabbar
Anagh Lal
Dejan Nickovic
Francesca Rossi
Peter Szilagyi
Shang-Wen Cheng

Table of Contents

Full Papers

Extended Abstracts

Invited Talks

Research Summaries

Verifying the Incorrectness of Programs and Automata*

Scot Anderson and Peter Revesz

Department of Computer Science and Engineering,
University of Nebraska-Lincoln, Lincoln, NE 68588, USA
{scot, revesz}@cse.unl.edu

Abstract. Verification of the incorrectness of programs and automata needs to be taken as seriously as the verification of correctness. However, there are no good general methods that always terminate and prove incorrectness. We propose one general method based on a *lower bound* approximation of the semantics of programs and automata. Based on the lower-bound approximation, it becomes easy to check whether certain error states are reached. This is in contrast to various abstract interpretation techniques that make an *upper bound* approximation of the semantics and test that the error states are not reached. The precision of our lower bound approximation is controlled by a single parameter that can be adjusted by the user of the MLPQ system in which the approximation method is implemented. As the value of the parameter decreases the implementation results in a finer program semantics approximation but requires a longer evaluation time. However, for all input parameter values the program is guaranteed to terminate. We use the lower bound approximation to verify the incorrectness of a subway train control automaton. We also use the lower bound approximation for a problem regarding computer security via trust management programs. We propose a trust management policy language extending earlier work by Li and Mitchell. Although, our trust management programming language is Turing-complete, programs in this language have semantics that lend themselves naturally to a lower-bound approximation. Namely, the lower bound approximation is such that no unwarranted authorization is given at any time, although some legitimate access may be denied.

1 Introduction

Testing the correctness of a program or an automaton can be done by finding an *upper approximation* of its semantics. If the upper approximation *does not* contain the error states needed to be checked, then the automaton can be said to be correct. However, if the upper approximation *contains* the error states, then the actual program or automaton may still be correct.

* This research was supported in part by NSF grant EIA-0091530 and a NASA Space and EPSCoR grant.

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 1–13, 2005.

Similarly, if the *lower bound* approximation of the semantics contains an error state, then we know that it is incorrect. If it does not, then the program may still be incorrect.

Hence an *upper bound* may be good to verify that a program is correct, while a *lower bound* may be good to verify that it is incorrect. The *verification of incorrectness* is just as important in practice as the verification of correctness, because many users are reluctant to change incorrect and expensive programs unless those are proven incorrect. For example, if a banking system allows invalid access to some bank accounts, then a lower bound approximation would be needed to verify the incorrectness.

Until recently, in the verification area the focus was in verifying correctness using *abstract interpretation* [8, 16, 22] or *model checking* [1, 5, 9, 30, 36]. In contrast, in this paper, we focus on verifying incorrectness.

Verifying incorrectness is needed when we suspect a program to be incorrect, and we want to prove that it is indeed incorrect. For example, if there is an accident with a space shuttle, then we need to find what caused it. Was it caused by an incorrect program?

There are many reasons that a program may be suspected to be incorrect. For example, a program that fails a verification for correctness using abstract interpretation or model checking would be suspicious.

There are some problems that naturally lend themselves to a lower-bound approximation. For example, the semantics of a computer security system would contain the facts that describe who gets access to which resource at what time. In this case a lower-bound approximation is meaningful, conservative, and safe to use. That is, it never gives unwarranted authorizations, although some legitimate access may be denied at certain time instances. For example, not being able to access one's own bank account at a particular time is frustrating, but it is certainly less frustrating than if someone else, who should not, can access it.

We use the above idea in proposing a Turing-complete extension of the *trust management* language RT [25, 26, 27], which is a recent approach to computer security in a distributed environment. The latest version of the RT language uses Datalog but with simpler constraints than we allow in this paper. We choose the RT trust management family of languages *as an example* of how to use constraint database approximation techniques in other areas beyond database systems where lower-bound approximations are meaningful. (See the survey [15] and the recent article [24] about trust management in general.)

The rest of this paper is organized as follows. Section 2 gives a brief review of constraint database approximation theory and its implementation in the MLPQ constraint database system [38]. Section 3 applies the approximation method to verify the incorrectness of an automaton. Section 4 applies the approximation method to find a safe evaluation of a trust management program. Section 5 discusses some related work. Finally, Section 6 gives some conclusions and future work.

2 Review of Constraint Database Approximation Theory

The *constraint logic programming* languages proposed by Jaffar and Lassez [17], whose work led to CLP(R) [19], by Colmerauer [7] within Prolog III, and by Dincbas et al. [10] within CHIP, were Turing-complete. Kanellakis, Kuper, and Revesz [20, 21] considered those to be impractical for use in database systems and proposed less expressive *constraint query languages* that have nice properties in terms of guaranteed and efficient evaluations. Many researchers advocated extensions of those languages while trying to keep termination guaranteed. For example, the least fixed point semantics of Datalog (Prolog without function symbols and negation) with integer gap-order constraint programs can be always evaluated in a finite constraint database representation [33].[1]

With gap-order constraints many NP-complete problems can be expressed that cannot be expressed in Datalog without constraints. However, even Datalog with addition constraints, which seems only a slight extension, is already Turing-complete. Hence Revesz [35] introduced an approximate evaluation for Datalog with addition constraints.

This approximation is different from *abstract interpretation* methods (for a recent review see [8]). The main difference is that, at least in theory, in [35] both a lower and an upper bound approximation of the least fixed point can be arbitrarily close to the actual least fixed point with the decrease of a single parameter towards $-\infty$. The decrease indirectly increases the running time.

Below we focus on the definitions that are relevant to approximations. The reader can find more details in the surveys [18, 34] and the books [23, 28, 37] about constraint logic programming and constraint databases.

Definition 1. *Addition constraints [37] have the form*

$$\pm x \pm y \ \theta \ b \quad or \quad \pm x \ \theta \ b$$

where x and y are integer variables and b is an integer constant, called a bound*, and θ is either $\geq$ or $>$.*

In the following we will also use $x = b$ as an abbreviation for the conjunction of $x \geq b$ and $-x \geq -b$. Similarly, we use $x + y = b$ as an abbreviation for the conjunction of $x + y \geq b$ and $-x - y \geq -b$.

Each *constraint database* is a finite set of *constraint tuples* of the form:

$$R(x_1, \ldots, x_k) \ :- \ C_1, \ldots, C_m.$$

where R is a k-ary relation symbol, each x_i for $1 \leq i \leq k$ is an integer variable or constant, and each C_j for $1 \leq j \leq m$ is an addition constraint over the variables. The meaning of a constraint tuple is that each substitution of the variables by integer constants that makes each C_j on the right hand side of $:-$ true is a k-tuple that is in relation R.

[1] A gap-order is a constraint of the form $x - y \geq c$ or $\pm x \geq c$ where x and y are variables and c is a non-negative integer constant.

A *Datalog program* consists of a finite set of constraint tuples and rules of the form:

$$R_0(x_1, \ldots, x_k) \ :- \ R_1(x_{1,1}, \ldots, x_{1,k_1}), \ldots, R_n(x_{n,1}, \ldots, x_{n,k_n}), \ C_1, \ldots, C_m.$$

where each R_i is a relation name, and the xs are either integer variables or constants, and each C_j is an addition constraint over the xs. The meaning of the rule is that if for some substitution of the variables by integer constants each R_i and C_j on the right hand side of $:-$ is true, then the left hand side is also true.

A *model* of a Datalog program is an assignment to each k-arity relation symbol R within the program a subset of $\mathbb{Z}^k$ where $\mathbb{Z}$ is the set of integers such that each rule holds for each possible substitution. The *least fixed point* semantics of a Datalog program contains the intersection of all the models of the program.

It is easy to express in Datalog [37] with addition constraints a program that will not terminate using a standard bottom-up evaluation [37]. Consider the following Datalog with addition constraint program:

$$\begin{array}{lll} D(x,y,z) & :\!- & x-y=0, \quad z=0. \\ D(x',y,z') & :\!- & D(x,y,z), \quad x'-x=1, \quad z'-z=1. \end{array} \tag{1}$$

This expresses that the *Difference* of x and y is z. Further, based on (1) we can also express a *Multiplication* relation as follows:

$$\begin{array}{lll} M(x,y,z) & :\!- & x=0, \quad y=0, \quad z=0. \\ M(x',y,z') & :\!- & M(x,y,z), \quad D(z',z,y), \quad x'-x=1. \\ M(x,y',z') & :\!- & M(x,y,z), \quad D(z',z,x), \quad y'-y=1. \end{array} \tag{2}$$

Intuitively, a standard bottom-up evaluation derives additional constraint tuples until a certain saturation is reached, and the saturation state represents in a constraint database form the least fixed point. We omit the precise definition of bottom-up evaluation of Datalog with constraint programs, because it is not needed for the rest of this paper. It is enough to note that the simple Datalog program that consists of the above two sets of rules never terminates in a standard bottom-up evaluation.

In fact, with these two relations we can express any integer polynomial equation (see Example 3). Since integer polynomial equations are unsolvable in general [29], no algorithm would be able to evaluate precisely the least fixed point semantics of the Datalog program. Hence the situation we face is not just a particular problem with the standard bottom-up evaluation, but a problem that is inherent to the least fixed point semantics of Datalog with addition constraints.

Revesz [35] introduced two methods for approximating the least fixed point evaluation by modifying the standard bottom-up evaluation.

Definition 2. *Let $l < 0$ be any fixed integer constant. We change in the constraint tuples the value of any bound b to be $max(b, l)$. Given a Datalog program P the result of a bottom-up evaluation of P using this modification is denoted P_l.*

Definition 3. *Let $l < 0$ be any fixed integer constant. We delete from each constraint tuple any constraint with a bound that is less than l. Given a Datalog program P the result of a bottom-up evaluation of P using this modification is denoted P^l.*

These modifications lead to the following approximation theorem.

Theorem 1. *[35] For any Datalog program P and constant $l < 0$ the following is true.*

$$P_l \subseteq lfp(P) \subseteq P^l$$

where $lfp(P)$ is the least fixed point of P. Further, P_l and P^l can be computed in finite time.

We can also get better and better approximations using smaller and smaller values of l. In particular, we have the following theorem.

Theorem 2. *[35] For any Datalog with addition constraints program P and constants l_1 and l_2 such that $l_1 \leq l_2 < 0$, the following hold:*

$$P_{l_2} \subseteq P_{l_1} \quad \text{and} \quad P^{l_1} \subseteq P^{l_2}$$

Because we are interested in evaluations that are lower bounds of the least fixed point $lfp(P)$, we implemented P_l as defined in Definition 2. The implementation was done within the MLPQ constraint database system [38], which is available from the website: `cse.unl.edu/~revesz`. The implementation is a new result that is not described in any other publication.

3 Verifying the Incorrectness of a Subway Automaton

We consider counter automata $\mathcal{A}$ which are tuples $(S, X, \tau, s_0, \overline{x}_0)$ where S is a finite set of states, X is a finite set of state counters $x_1, \ldots, x_k$ which are integer variables, τ is a finite set of transitions from S to S, s_0 is an initial state, and $\overline{x}_0$ is an initial assignment of the state variables. Each transition has two parts, a *guard constraint* over the variables that needs to be satisfied before the transition takes place and a set of assignments to the variables that update their values as the automaton enters the new state. In this paper we allow only addition constraints in the guard constraints and assignments that can be expressed by addition constraints.

Counter machines are an example of such automata which allow only guard constraints that are comparisons between variables and constants and assignments that increment and decrement a variable by one or set a variable to a constant. They were studied by Minsky [31, 32], who showed that they have the same expressive power as Turing machines. Floyd and Beigel [11] is an introduction to automata theory that covers counter machines.

More complex guard constraints have been allowed in later extensions of counter machines and applied to the design of control systems in Boigelot and

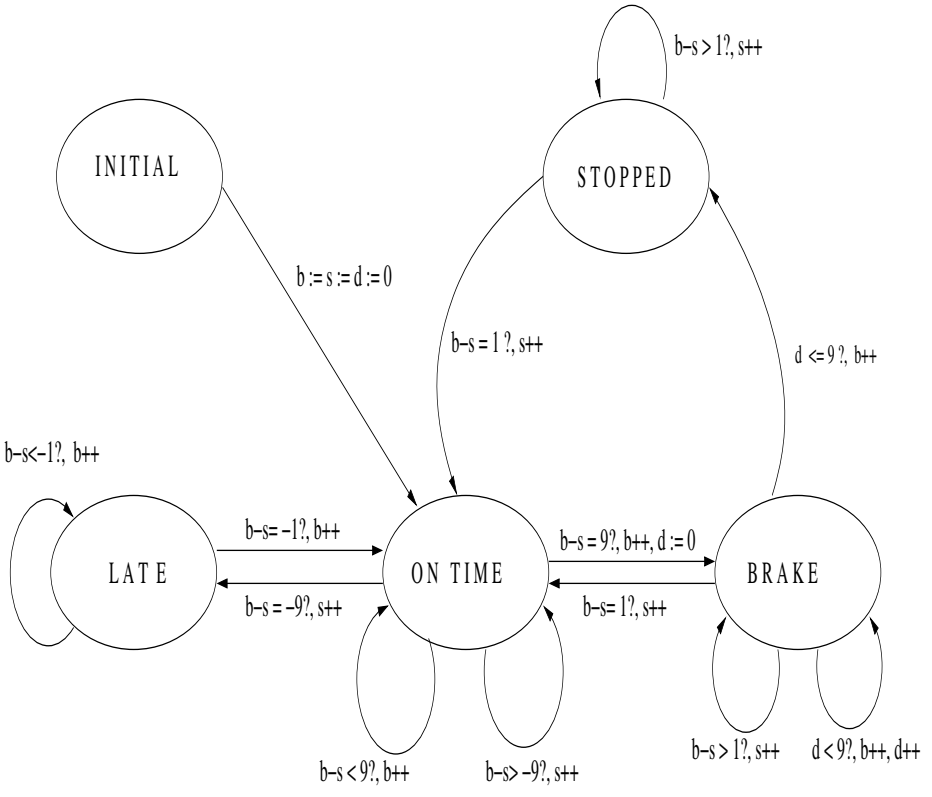


Fig. 1. The subway train control system

Wolper [4], Fribourg and Olson [12], Fribourg and Richardson [13], Halbwachs [16], and Kerbrat [22]. Boigelot et al. [3], Cobham [6], and Wolper and Boigelot [39] study automata and Presburger definability. For additional discussion and examples of various types of counter (constraint) automata see [37].

Let us consider the following subway train speed regulation system described by Halbwachs [16]. Each train detects beacons that are placed along the track and receives a "second" signal from a central clock.

Let b and s be counter variables for the number of beacons and second signals received. Further, let d be a counter variable that describes how long the train is applying its brake. The goal of the speed regulation system is to keep $\mid b - s \mid$ small while the train is running.

The speed of the train is adjusted as follows. When $s + 10 \leq b$, then the train notices it is early and applies the brake as long as $b > s$. Continuously braking causes the train to stop before encountering 10 beacons.

When $b + 10 \leq s$ the train is late and will be considered late as long as $b < s$. As long as any train is late, the central clock will not emit the second signal.

The subway speed regulation system can be drawn as a constraint automaton shown in Figure 3, where the guard constraints are followed by question marks, and $x++$ and $x--$ are abbreviations for the assignments $x := x + 1$ and $x := x - 1$, respectively, for any variable x.

The set of reachable configurations (combinations of states and state variable values) of the automaton shown in Figure 3 can be expressed in Datalog with addition constraints by creating a new ternary relation for each state with the order of variables (b, d, s) and writing the following Datalog with addition constraints rules:

$$
\begin{array}{lcl}
Brake(b, s', d) & :\!- & Brake(b, s, d),\ \ b - s > 1,\ \ s' - s = 1. \\
Brake(b', s, d') & :\!- & Brake(b, s, d),\ \ -d > -9,\ \ b' - b = 1,\ \ d' - d = 1. \\
Brake(b', s, d') & :\!- & Ontime(b, s, d),\ \ b - s = 9,\ \ b' - b = 1,\ \ d' = 0. \\
 & & \\
Initial(b, s, d) & :\!- & b = 0,\ \ s = 0,\ \ d = 0. \\
 & & \\
Late(b', s, d) & :\!- & Late(b, s, d),\ \ -b + s > 1,\ \ b' - b = 1. \\
Late(b, s', d) & :\!- & Ontime(b, s, d),\ \ b - s = -9,\ \ s' - s = 1. \\
 & & \\
Ontime(b, s', d) & :\!- & Brake(b, s, d),\ \ b - s = 1,\ \ s' - s = 1. \\
Ontime(b, s, d) & :\!- & Initial(b, s, d). \\
Ontime(b', s, d) & :\!- & Late(b, s, d),\ \ b - s = -1,\ \ b' - b = 1. \\
Ontime(b', s, d) & :\!- & Ontime(b, s, d),\ \ -b + s > -9,\ \ b' - b = 1. \\
Ontime(b, s', d) & :\!- & Ontime(b, s, d),\ \ b - s > -9,\ \ s' - s = 1. \\
Ontime(b, s', d) & :\!- & Stopped(b.s.d),\ \ b - s = 1,\ \ s' - s = 1. \\
 & & \\
Stopped(b', s, d) & :\!- & Brake(b, d, s),\ \ -d \geq -9,\ \ b' - b = 1. \\
Stopped(b, s', d) & :\!- & Stopped(b, s, d),\ \ b - s > 1,\ \ s' - s = 1.
\end{array}
$$

Error Condition: Suppose that this automaton is correct if $|b - s| < 20$ in all states at all times. Then this automaton is incorrect if $|b - s| \geq 20$ at least in one state at one time. The table below shows the result of the lower bound approximation using the MLPQ constraint database system.

MLPQ Lower-Bound

Brake	Late	Ontime	Stopped
$1 \leq b - s \leq 19$	$-10 \leq b - s \leq -1$	$-9 \leq b - s \leq 9$	$1 \leq b - s \leq 20$
$10 \leq b \leq 19$	$10 \leq s \leq 19$	$0 \leq b \leq 9$	$11 \leq b \leq 20$
$0 \leq s \leq 18$	$0 \leq d \leq 9$	$0 \leq s \leq 18$	$0 \leq s \leq 9$
$0 \leq d \leq 9$		$0 \leq d \leq 9$	$0 \leq d \leq 9$

The above was obtained by using $l = -30$ as in Definition 2. If l is decreased, then the upper bounds of b and s increase in the above table. Therefore, in the limit those upper bounds can be dropped.

Further, for any value of u, since the above is a lower bound, any possible integer solution of the constraints below the state names *must occur* at some

time. For example, the *Stopped* state must contain the case $b - s = 20$ at some time. Therefore, this automaton is incorrect by our earlier assumption.

3.1 Comparison with Verimag

The Verimag laboratory has software for testing program correctness using abstract interpretation. Halbwachs [16] gave the following upper bound derived using Verimag's software for the subway automaton.

Verimag Upper-Bound

Brake	Late	Ontime	Stopped
$1 \leq b - s \leq d + 10$	$-10 \leq b - s \leq -1$	$-9 \leq b - s \leq 9$	$1 \leq b - s \leq 19$
$d + 10 \leq b$	$s \geq 10$	$b \geq 0$	$19 \leq 9s + b$
$0 \leq d \leq 9$		$s \geq 0$	$b \geq 10$

Surprisingly, this result does not match our result. In particular, the upper bound for the *Stopped* state contains the constraint $b - s \leq 19$, which says that the value of $b - s$ cannot be 20, but our lower bound says that 20 must be one of the cases. To resolve this apparent contradiction, we need to look more closely at the automaton. We can see that the following is a valid sequence of transitions, where $S(b, s, d)$ represents the values of b, s, and d is each state $S \in \{Brake, Initial, Late, Ontime, Stopped\}$.

$Initial(0,0,0) \longrightarrow Ontime(0,0,0) \longrightarrow Ontime(1,0,0). \longrightarrow Ontime(2,0,0) \longrightarrow$
$Ontime(3,0,0) \longrightarrow Ontime(4,0,0) \longrightarrow Ontime(5,0,0) \longrightarrow Ontime(6,0,0) \longrightarrow$
$Ontime(7,0,0) \longrightarrow Ontime(8,0,0) \longrightarrow Ontime(9,0,0) \longrightarrow Brake(10,0,0) \longrightarrow$
$Brake(11,0,1) \longrightarrow Brake(12,0,2) \longrightarrow Brake(13,0,3) \longrightarrow Brake(14,0,4) \longrightarrow$
$Brake(15,0,5) \longrightarrow Brake(16,0,6) \longrightarrow Brake(17,0,7) \longrightarrow Brake(18,0,8) \longrightarrow$
$Brake(19,0,9) \longrightarrow Stopped(20,0,9)$

Note that $Stopped(20,0,9)$ contradicts the first constraint in the Verimag upper bound for the *Stopped* state. Hence we suspect that the Verimag software contains some bug or there was some problem in data entry. We suggest that its incorrectness be tested using other examples and our lower-bound method.

4 Approximating Trust Management Program Semantics

Trust management languages allow the expression of high-level rules about which principal can get access to which resource at what time in a distributed environment. The *Keynote* trust management system [2, 25] allowed integer polynomial constraints. However, later trust management systems do not allow such constraints, because allowing them leads to undecidability [29].

We argue that this restriction unnecessarily limits the expressibility of trust management languages. Our lower bound method can be used in most cases to

verify the correctness of an access even when the rules contain integer polynomial constraints.

Note that an upper bound approximation may allow some access which is not specified by the trust management rules. Hence it is not appropriate for trust management, while a lower bound technique can be safely used. In a computer security system it leads to much less harm if a legitimate access request is denied (which can happen with a lower bound approximation) than if an illegitimate access is allowed (which can happen with an upper bound approximation).

Integer polynomial constraints arise naturally in security applications, as shown by the following example.

Example 1. Suppose an e-mail sender or server organization C needs to assign a level of trust to an individual based on the trust levels assigned by organizations A and B. Suppose C considers B's information much more valuable. Then C may use the following integer polynomial constraint to assign a trust level of its own:

$$3Level_C \ \geq \ 4(Level_A)^2 \ + \ 2Level_B \tag{3}$$

4.1 Extended RT Syntax

RT is a trust management policy language introduced by Li et al. [27]. The parameters in each of the different kinds of policy statements in RT define the relationships between *principal owners*, the *roles* they own and the *members* of the roles.

The following *simple member* rule defines the principal K_D to be a member of role R owned by K_A:

$$K_A.R(p_1, ..., p_n) \ \leftarrow \ K_D \tag{4}$$

where the role takes the form $R(p_1, ..., p_n)$, and R is a role name and each p_j is a variable in an order constraint.

Extended RT: We extend the RT language by allowing each p_j to be an integer variable within an integer polynomial constraint. The *extended simple member* rules have the syntax:

$$K_A.R(p(x_1, ..., x_l)) \ \leftarrow \ K_D \tag{5}$$

where K_A defines a role R to contain member K_D, if the integer polynomial constraint $p(x_1, ..., x_l)$ holds.

Example 2. The extended *RT* statement

$$Email.Permit(3Level_C - 4(Level_A)^2 - 2Level_B \geq 0) \leftarrow "Charlie" \tag{6}$$

allows Charlie access to e-mail, if the ratings obtained satisfy constraint (3).

4.2 Extended RT Semantics

The semantics of an extended RT program can be found by translating the extended RT rules into logically equivalent Datalog with addition constraints rules and then taking the least fixed point semantics of the resulting Datalog program.

Extended simple member rules of form (4) are translated into the following Datalog rule:

$$R(K_A, K_D, x_1, ..., x_m) \ :- \ p_1, ..., p_n. \tag{7}$$

where $x_1, ..., x_m$ are the integer constants and variables that may be used within the polynomial constraints $p_1, ..., p_n$.

We can translate integer polynomial constraints with k number of $+$ and $\times$ operations into a conjunction of at most $2k$ difference D and multiplication M relations defined in Section 2.

Example 3. The extended *RT* statement (6) can be translated into the following Datalog with addition constraints rule:

$$\begin{aligned} Permit(Email, Charlie, Level_A, Level_B, Level_C) :- \ & M(t_1,\ 3,\ Level_C), \\ & M(t_2,\ Level_A,\ Level_A), \\ & M(t_3,\ 4,\ t_2), \\ & M(t_4,\ 2,\ Level_B), \\ & D(e_1,\ t_1,\ t_3), \\ & D(e_2,\ e_1,\ t_4), \\ & e_2 \geq 0 \end{aligned}$$

where $Level_A, Level_B$ and $Level_C$ are either integer variables or constants and are the important parameters in this problem, while each t_i and e_1 and e_2 are additional integer variables that are introduced only for the sake of expressing the polynomial equation. Finally, *Email* and *Charlie* are integer constants that represent the strings "Email" and "Charlie" in the RT statement (6).

Since the difference D and multiplication M relations have already been defined in Section 2 using Datalog with addition constraints, the entire Datalog program can be evaluated using the lower-bound approximation of its least fixed point by a modified bottom-up evaluation. By Theorem 1 this evaluation terminates, giving a lower bound of the semantics of the extended RT program.

5 Related Work

There are few papers on lower bounds for automata and programs. Godefroid et al. [14] gives a lower-bound approximation of the automaton by simplifying its states according to some predicates that hold in each state. The state transitions considered in the simplification are must-transitions, that is, if the condition in the previous state holds, then only one subsequent state can be reached. Unfortunately, this is very limited, because in fact most transitions among states are

not must-transitions. In general, predicate abstraction methods, such as [14], can yield more precise approximations with the introduction of additional predicates, making the automata structures increasingly more complex.

In contrast, our approximation is radically different and does not change at all the automata structure, rather it indirectly changes the algebra in which (polynomial) constraints are interpreted and solved. Essentially, the simplified algebra relies on modified addition and multiplication relations. These relations are smoothly and naturally extended as l decreases. Hence our method may yield more precise approximations without increasing the size of the automaton.

6 Conclusions and Future Work

We have seen that in general decreasing the bound l toward $-\infty$ leads to tighter lower and upper bound approximations, P_l and P^l, respectively. If the lower and upper bound approximations agree (i.e, $P_l = P^l$), then we know that we have found the precise least fixed point. However, even if they do not agree, but seem to converge to the same value –and that may be apparent from only a few examples of l values, then we still can give the limit of convergence as the precise least fixed point. The subtle point is that the series of lower (upper) bound approximations themselves show a convergence and hence their limits can be approximated. It is an approximation of approximations, but it may work beautifully in many cases.

Open Problem: Determine the precise conditions under which the least fixed point can be predicted as described above.

We have to be cautious not to overclaim the potential of the above approach, for it is easy to see that the above method may fail sometimes. For example, consider any query that requires a polynomial integer equation that is build using the `Diff` and `Mult` relations. Clearly, the solutions of the polynomial equation can be found using the `Diff` and `Mult` relations built using lower or upper bound approximation. However, decreasing l does not guarantee finding tighter lower and upper bound approximations for the polynomial equation. It may be impossible to tell when all the solutions will be found. Since integer polynomial equations are undecidable in general, there always will be some cases when the convergence is unpredictable.

Considering a general difference constraint, if the upper bound approximation is infinite, then it may not be possible to predict how the two bounds approach one another. In fact the lower bound may grow without bound as l approaches $-\infty$.

Hence in general we conclude by the examples above that there exists a class of queries that are not stable. We also conjecture that when $|P^l \backslash P_l| = \infty$ and the expected solution is finite the query is not stable.

In the future, we would like to determine which classes of queries are stable and implement routines that predict an accurate solution to improve the approximation process when enough evidence is collected to make a precise prediction

of convergence. That would result in a kind of approximation that is neither a simple lower nor a simple upper bound approximation but is something much more sophisticated.

In conclusion,

References

1. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T., Ho, P.-H., Nicollin, X., Olivero, A., Sifakis, J., and Yovine, S. The algorithmic analysis of hybrid systems. *Theoretical Computer Science 138*, 1 (1995), 3–34.
2. Blaze, M., Feigenbaum, J., and Lacy, J. Decentralized trust management. Tech. Rep. 96-17, AT and T Research, 1996.
3. Boigelot, B., Rassart, S., and Wolper, P. On the expressiveness of real and integer arithmetic automata. In *International Colloquium on Automata, Languages and Programming* (1998), vol. 1443 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 152–63.
4. Boigelot, B., and Wolper, P. Symbolic verification with periodic sets. In *Proc. Conference on Computer-Aided Verification* (1994), pp. 55–67.
5. Clarke, E. M., Grumberg, O., and Peled, D. A. *Model Checking*. MIT Press, 1999.
6. Cobham, A. On the base-dependence of sets of numbers recognizable by finite automata. *Mathematical Systems Theory 3* (1969), 186–92.
7. Colmerauer, A. Note sur Prolog III. In *Proc. Séminaire Programmation en Logique* (1986), pp. 159–174.
8. Cousot, P. Proving program invariance and termination by parametric abstraction, lagrangian relaxation and semidefinite programming. In *Sixth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'05)* (Paris, France, LNCS 3385, Jan. 17–19 2005), Springer, Berlin, pp. 1–24.
9. Delzanno, G., and Podelski, A. Model checking in CLP. In *2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (1999), vol. 1579 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 74–88.
10. Dincbas, M., Van Hentenryck, P., Simonis, H., Aggoun, A., Graf, T., and Berthier, F. The constraint logic programming language chip. In *Proc. Fifth Generation Computer Systems* (Tokyo, Japan, 1988), pp. 693–702.
11. Floyd, R. B., and Beigel, R. *The Language of Machines: An Introduction to Computability and Formal Languages*. Computer Science Press, 1994.
12. Fribourg, L., and Olsén, H. A decompositional approach for computing least fixed-points of datalog programs with Z-counters. *Constraints 2*, 3–4 (1997), 305–36.
13. Fribourg, L., and Richardson, J. D. C. Symbolic verification with gap-order constraints. In *Proc. Logic Program Synthesis and Transformation* (1996), vol. 1207 of *Lecture Notes in Computer Science*, pp. 20–37.
14. Godefroid, P., Huth, M., and Jagadeesan, R. Abstraction-based model checking using modal transition systems. In *12th International Conference on Concurrency Theory* (2001), pp. 426–440.
15. Grandison, T., and Sloman, M. A survey of trust in internet application. *IEEE Communications Surveys and Tutorials 3*, Fourth Quarter (2000).
16. Halbwachs, N. Delay analysis in synchronous programs. In *Proc. Conference on Computer-Aided Verification* (1993), pp. 333–46.

17. Jaffar, J., and Lassez, J. L. Constraint logic programming. In *Proc. 14th ACM Symposium on Principles of Programming Languages* (1987), pp. 111–9.
18. Jaffar, J., and Maher, M. Constraint logic programming: A survey. *J. Logic Programming 19/20* (1994), 503–581.
19. Jaffar, J., Michaylov, S., Stuckey, P. J., and Yap, R. H. The CLP(R) language and system. *ACM Transactions on Programming Languages and Systems 14*, 3 (1992), 339–95.
20. Kanellakis, P. C., Kuper, G. M., and Revesz, P. Constraint query languages. In *Proc. ACM Symposium on Principles of Database Systems* (1990), pp. 299–313.
21. Kanellakis, P. C., Kuper, G. M., and Revesz, P. Constraint query languages. *Journal of Computer and System Sciences 51*, 1 (1995), 26–52.
22. Kerbrat, A. Reachable state space analysis of lotos specifications. In *Proc. 7th International Conference on Formal Description Techniques* (1994), pp. 161–76.
23. Kuper, G. M., Libkin, L., and Paredaens, J., Eds. *Constraint Databases*. Springer-Verlag, 2000.
24. Li, N., and Mitchell, J. Understanding SPKI/SDSI using first-order logic. In *Proc. IEEE Computer Security Foundations Workshop* (2003), pp. 89–108.
25. Li, N., and Mitchell, J. C. Datalog with constraints: A foundation for trust management languages. In *Proceedings of the Fifth International Symposium on Practical Aspects of Declarative Languages* (Jan. 2003), pp. 58–73.
26. Li, N., and Mitchell, J. C. RT: A role-based trust-management framework, April 2003.
27. Li, N., Mitchell, J. C., and Winsborough, W. H. Design of a role-based trust management framework. In *Proc. IEEE Symposium on Security and Privacy, Oakland* (May 2002).
28. Marriott, K., and Stuckey, P. J. *Programming with Constraints: An Introduction*. MIT Press, 1998.
29. Matiyasevich, Y. Enumerable sets are diophantine. *Doklady Akademii Nauk SSR 191* (1970), 279–82.
30. McMillan, K. *Symbolic Model Checking*. Kluwer, 1993.
31. Minsky, M. L. Recursive unsolvability of Post's problem of "tag" and other topics in the theory of Turing machines. *Annals of Mathematics 74*, 3 (1961), 437–55.
32. Minsky, M. L. *Computation: Finite and Infinite Machines*. Prentice Hall, 1967.
33. Revesz, P. A closed-form evaluation for Datalog queries with integer (gap)-order constraints. *Theoretical Computer Science 116*, 1 (1993), 117–49.
34. Revesz, P. Constraint databases: A survey. In *Semantics in Databases*, L. Libkin and B. Thalheim, Eds., vol. 1358 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998, pp. 209–46.
35. Revesz, P. Datalog programs with difference constraints. In *Proc. 12th International Conference on Applications of Prolog* (1999), pp. 69–76.
36. Revesz, P. Reformulation and approximation in model checking. In *Proc. 4th International Symposium on Abstraction, Reformulation, and Approximation* (2000), B. Choueiry and T. Walsh, Eds., vol. 1864 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 124–43.
37. Revesz, P. *Introduction to Constraint Databases*. Springer-Verlag, 2002.
38. Revesz, P., Chen, R., Kanjamala, P., Li, Y., Liu, Y., and Wang, Y. The MLPQ/GIS constraint database system. In *ACM SIGMOD International Conference on Management of Data* (2000).
39. Wolper, P., and Boigelot, B. An automata-theoretic approach to Presburger arithmetic constraints. In *Proc. Static Analysis Symposium* (1995), vol. 983 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 21–32.

Generating Admissible Heuristics by Abstraction for Search in Stochastic Domains

Natalia Beliaeva and Shlomo Zilberstein

Department of Computer Science,
University of Massachusetts Amherst, USA
{nbeliaev, shlomo}@cs.umass.edu

Abstract. Search in abstract spaces has been shown to produce useful admissible heuristic estimates in deterministic domains. We show in this paper how to generalize these results to search in stochastic domains. Solving stochastic optimization problems is significantly harder than solving their deterministic counterparts. Designing admissible heuristics for stochastic domains is also much harder. Therefore, deriving such heuristics automatically using abstraction is particularly beneficial. We analyze this approach both theoretically and empirically and show that it produces significant computational savings when used in conjunction with the heuristic search algorithm LAO*.

1 Introduction

The Markov decision process (MDP) is widely used in artificial intelligence to solve problems of planning and learning under uncertainty. The most common way to solve an MDP is by using a dynamic programming algorithm such as value iteration or policy iteration. The major drawback of this approach is that the entire state space has to be evaluated. More recently, heuristic search algorithms have been developed for solving MDPs [6]. These algorithms can avoid evaluating states that are not reachable from the start state by an optimal policy. The effectiveness of heuristic search mostly depends on the heuristic function being used to guide the search process. One way to generate an *admissible* heuristic is to use search in abstract spaces [7, 11]. Abstraction works by replacing an original state space by an abstract space, which is easier to search. This idea is not new. It has been previously applied to creating admissible heuristics for A* search [8]. More recently, there has been growing interest in developing heuristics using a form of abstraction called *pattern database* [3, 9, 10]. The goal of this paper is to extend the use of abstraction as a means of creating admissible heuristics for search in stochastic domains. Heuristic estimates generated by abstraction are then used to guide LAO*, which is a heuristic search algorithm that can be used to solve stochastic planning problems. To test whether heuristics generated by abstraction produce any savings as compared to uninformed search, LAO* algorithm is applied to a task planning problem that involves uncertainty regarding the use of resources. The structure of this problem facilitates the creation of an abstract space very easily by varying the resolution of resource usage, always rounding up the amount of resources left for future activity.

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 14–29, 2005.

We show that heuristic estimates obtained by search in such abstract space are always admissible. That is, the heuristic value is an optimistic estimate (overestimate) of the actual value of a state. We also show that the effectiveness of such heuristic estimates depends on the problem and that they could result in significant savings compared to blind search.

The rest of the paper is organized as follows. In Section 2, we review some related work on search in abstract spaces and alternative approximation techniques for MDPs. Section 3 describes the general methodology used in the current research. Section 4 describes the specific model used in the paper. Section 5 analyzes experimental results. Section 6 concludes the paper with a summary of contributions and further work.

2 Related Work

We describe briefly related work in two research areas. First, we examine previous work on the problem of creating heuristics by abstraction in deterministic settings. This paper extends this body of research to stochastic domains. We then describe existing exact and approximate techniques for solving MDPs; heuristic search presents an alternative approach to these techniques.

2.1 Creating Heuristic by Abstraction for Search in Deterministic Domains

Several researchers have looked at the problem of creating heuristic by abstraction for search in deterministic domains (for example, [7, 8, 11, 12]). The most relevant work to the current study is Holte *et al.* [8]. This paper focuses on one type of abstraction called homomorphism (grouping together states of the original state space to create a single abstract state). The heuristic created by abstraction is then used to guide A* search. The goal of the paper was to develop a technique that would break Valtorta's barrier. To achieve this goal the number of states expanded by a heuristic search has to be less than the number of states expanded by uninformed search. The authors use an abstraction-based search algorithm called hierarchical A*. To create an abstraction they use the STAR abstraction technique, which groups together neighboring states within a certain radius. Once one level of abstraction is created, the procedure is repeated recursively until a trivial abstract level is created. The implementation of hierarchical A* is standard except for the way heuristic values are estimated. Every time A* needs a heuristic estimate, it is computed by searching at the next level of abstraction. It was found that a naive version of the algorithm ends up expanding many more states as compared to uninformed search, i.e. Valtorta's barrier is not broken. This could be explained by the fact that although A* never expands the same state twice in a single search, it has to expand the same state many times while performing multiple searches of the abstract levels. To overcome this problem the authors implemented two types of cashing techniques and as a result the Valtorta's barrier was broken in every domain. The authors have also discovered that as the radius of abstraction increases, the number of nodes expanded by hierarchical A* decreases until it reaches some minimum value. Increasing the abstraction radius further caused the number of expanded nodes to increase. In every case the

best abstraction radius represented a large fraction of the search space and as a result the abstraction hierarchy contained only one non-trivial level. In this paper we examine the applicability of these same ideas in stochastic search and evaluate the effectiveness of the approach.

More recently, an effective approach to exploit abstraction in the form of a pattern database has been developed. The idea was introduced by Culberson and Schaeffer [3] who applied it to permutation problems, like the 15-puzzle. To form a pattern database, a search space is projected into an abstract space, which is small enough to allow an efficient computation of the value function for each abstract state. The computed values are stored in a look-up table. Each abstract state is called a pattern and the table that stores the optimal values is called a pattern database. These precomputed values are then used as heuristic estimates for the search in the original state space. Usually more than one pattern database is defined for the same problem. Heuristic estimates for the states of the original space are computed as maximum of several pattern database heuristics. For example, Korf defined three pattern databases to solve the Rubik's cube problem [10]. Similarly, Korf and Felner used eight pattern databases to solve the 24-puzzle [9].

2.2 Approximate Solutions to MDPs

A common way to solve MDPs is by using dynamic programming techniques such as value iteration or policy iteration. The problem with this approach is that the entire state space–which grows exponentially with the number of state variables–has to be evaluated. This makes it hard to find exact solutions, leading to a vast literature on techniques that can approximate the optimal solution. Both planning and learning techniques for approximation of MDP solutions have been developed. The goal of both planning and learning under uncertainty is to discover an optimal or near-optimal policy of action, represented as a mapping from states to actions. The main difference is that planning problems assume that the action model and the reward function are known, whereas learning problems assume both of these to be initially unknown and attempt to learn them. While planning is typically performed off-line, learning algorithms are frequently designed for on-line operation.

A large body of research that attempts to find an approximate solution to an MDP in the context of planning deals with reducing the level of detail in the problem representation by aggregating states with similar or identical values and/or action choices. These aggregate states are then treated as a group by the dynamic programming algorithm (see [5]). Another way to reduce the complexity of the problem is by pruning the tree representation of value functions by removing such nodes in the tree that induce small differences in value (see [2]) or by substituting the values at the terminals with ranges of values (see [13]). Another class of approximation procedures used in planning under uncertainty involves searching local regions or so called envelopes of the state space (see [4, 14]).

Solving MDPs has also been a focus area in reinforcement learning (RL). Most RL algorithms adapt dynamic programming algorithms so that they could be used on-line. To avoid the curse of dimensionality, many methods have been proposed to approximate MDP solutions. Barto and Mahadevan, for example, identify the following three methods for finding approximate solutions using RL algorithms [1]: (1) Restricting com-

putation to states along sample trajectories to avoid the exhaustive sweeps of dynamic programming; (2) Sampling from the appropriate distribution to simplify the basic dynamic programming backup; and (3) Representing the value function and/or policies more compactly by using function approximation methods, such as linear combinations of basic functions or neural networks.

The technique we present in this paper is an exact algorithm, but it can be easily transformed into an approximation technique. In previous work, we have shown how to convert any exact heuristic search algorithm into a "well-behaved" anytime algorithm that could produce approximate solutions with error-bounds that improve with computation time. However, examining the anytime characteristics of hierarchical LAO* is beyond the scope of this paper.

3 Methodology

3.1 The LAO* Algorithm

The LAO* algorithm was developed by Hansen and Zilberstein [6] as a heuristic approach to finding optimal solutions to MDPs. What distinguishes LAO* from other classical heuristic search algorithms, such as AO*, is the fact that it allows to find solutions that contain loops. Since LAO* is a heuristic search algorithm, it can avoid evaluating the entire search space which makes it a good alternative to dynamic programming algorithms that are commonly used to solve MDPs. Since LAO* does not evaluate every state of the problem, it is not necessary to supply the entire graph to the algorithm. Instead, a graph is specified implicitly by a start state and a successor function.

For complete details of the implementation of LAO* see Hansen, Zilberstein [6]. Generally, the algorithm has two main steps: a forward search step and a dynamic programming step. The forward step identifies and expands the best partial solution graph. The dynamic programming step updates the evaluation function and marks best action for each state that belongs to the current best solution. Although LAO* works correctly independently of which state of the best partial solution is expanded next, the performance of the algorithm can be improved by a good heuristic function. One way to construct a heuristic is by search in abstract spaces.

3.2 Heuristic Construction by Abstraction

Heuristics are designed to speed up search. However, construction of a good heuristic usually comes at a cost. The goal is to come up with a heuristic such that the cost of computing it is less than the savings from using it. The use of heuristic h is said to be beneficial if the total number of states expanded by search with heuristic h is less than the number of states expanded by "blind" or uninformed search. In a stochastic setting "blind" search is also equivalent to reachability analysis.

There are two types of abstraction that can be used to construct a heuristic:

1. Embedding – relaxing a problem by "adding edges" to a state space (for example, by dropping preconditions from, or adding macro-operators to the state-space definition).

2. Homomorphism – grouping together several states in the original state space to create a single state in the abstract space.

It was proven by Valtorta (see [15]) that A* search using heuristic constructed by embedding transformation cannot be beneficial. Holte *et al.* [8] have generalized Valtorta's theorem to any abstraction transformation. They have shown that if the abstraction used to direct A* is a homomorphism, then it can be beneficial. The potential savings are due to the fact that expansion of many states in the original space can be replaced by an expansion of a single state in the abstract space. The goal of this research is to see whether the same idea holds in a stochastic setting, i.e. whether an admissible heuristic can be constructed by homomorphism and whether it can be beneficial if used to direct LAO* search. The next section generalizes Holte *et al.* version of Valtorta's theorem to stochastic search spaces.

3.3 Valtorta's Theorem Generalized to Stochastic Search

Let *SP* be the original state space, *SP'* the abstraction of *SP*. Let S be any state necessarily expanded when the given problem (S_0, G) is solved by reachability analysis directly in space *SP*. Let f be any abstraction mapping from *SP* to *SP'* and $h_f(S)$ be computed by reachability analysis in *SP'* from $f(S)$ to $f(G)$. If the problem is solved in *SP* by LAO* search using $h_f(\cdot)$ as heuristic estimate, then either:

1. S itself will be expanded, or
2. $f(S)$ will be expanded

Proof. When LAO* terminates, every state will either be

1. expanded,
2. visited, or
3. or unvisited.

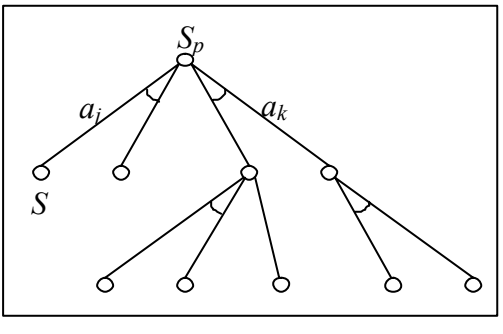


Fig. 1. State S_p is expanded and action a_k is chosen as the best action. State S is visited but not expanded

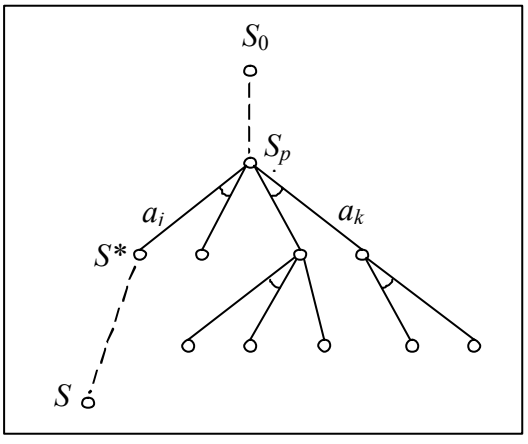


Fig. 2. State S^* is visited but not expanded. State S is unvisited

We examine each one of these cases below:

1. In the first case, the state S itself is expanded.
2. In the second case, the parent node S_p of state S, is expanded and the best action for S_p is computed (see Fig. 1). Because S is not expanded, the action a_j which leads to state S with a certain probability is suboptimal. However, to pick an optimal action for state S_p, $h_f(S)$ must have been computed. To compute $h_f(S)$, it is necessary to solve a problem $(f(S), f(G))$ in the abstract space by reachability analysis. Therefore, $f(S)$ has to be expanded at the first step.
3. In the third case, the state S is unvisited (see Fig. 2). It means that on every path from S_0 to S there must be a state which was visited but not expanded. Let S^* be such state on any shortest path from S_0 to S. As in the previous case, $h_f(S^*)$ must have been computed. To compute $h_f(S^*)$, it is necessary to solve a problem $(f(S^*), f(G))$ in the abstract space by reachability analysis. Since state S is reachable in the original state space, the corresponding state, $f(S)$. in the abstract space has to be reachable as well. Therefore, while solving the problem $(f(S^*), f(G))$ by reachability analysis, the state $f(S)$ has to be expanded.

3.4 General Problem Description

One type of problems that can be solved using LAO* with heuristics created by abstraction is executing multiple tasks that involve uncertainty about resources. In such problems, an agent has to perform a series of tasks. Every task is associated with a set of actions that an agent can undertake to complete the task. Each action uses an uncertain amount of one or several resources (for example, time, energy, etc.) and compensates the agent with a certain reward. The process stops once the agent either performs all tasks or runs out of at least one of the resources. The goal is to maximize the collected

reward while performing a sequence of tasks. The structure of such problems allows creating an abstract space very easily by grouping states by resources.

Each state is defined by the amount of resources left:

$$R_i = \{0,\ 1,\ 2, \ldots, N_i\}\,,\quad i = 1, \ldots, n$$

and by values of the variables:

$$V_i = \{1,\ 2, \ldots, M_i\}\,,\quad i = 1, \ldots, m$$

Formally, $S = \{R_1, \ldots, R_n;\ V_1, \ldots, V_m\}$. The start state can be defined as

$$S_0 = \{N_1, \ldots, N_n;\ 1, \ldots, 1\}$$

There are many possible terminal states. One example of a terminal state is

$$G = \{0, \ldots, R_n;\ V_1, \ldots, V_m\}$$

3.5 Creating an Abstract Space

To create the abstract space, it is first necessary to choose the desired granularity of abstraction or *abstraction step*. Selecting larger steps for grouping resources will result in a smaller total number of states in an abstract space and therefore fewer states to expand while performing a "blind" search. On the other hand, the generated heuristic estimates will be coarser and subsequently more states will need to be expanded in the original space. If smaller abstraction steps are used, then there will be more work in the abstract space, and less in the original. Therefore, it is important to use such abstraction steps that result in an optimal trade-off between the number of states expanded in the abstract and the original spaces.

When an abstract space is created, states of the original space are grouped by resources with each resource rounded up. Only states with the same variable values can be grouped. Since we are representing the amount of remaining resources (as opposed to the amount of used resources), rounding resources up ensures admissibility of the heuristic as it will always be overestimating the reward. Given that the amount of resources available in each state is overestimated, it might be possible to do more work (i.e. take more actions) and subsequently collect a larger reward.

4 The Model

4.1 Problem Specification

To test whether heuristic estimates generated by abstraction produce any savings as compared to uninformed search, the LAO* algorithm was used to solve the following problem that involves uncertainty regarding the use of resources. An agent operates autonomously for a period of time. Its goal is to perform a sequence of M tasks (see Fig. 3). A terminal state is reached when the agent either performs all tasks or runs out of at least one resource.

Each task can be executed either by taking an action *A*, or by taking an action *B*. In addition, some tasks may be skipped altogether. When a problem instance is created,

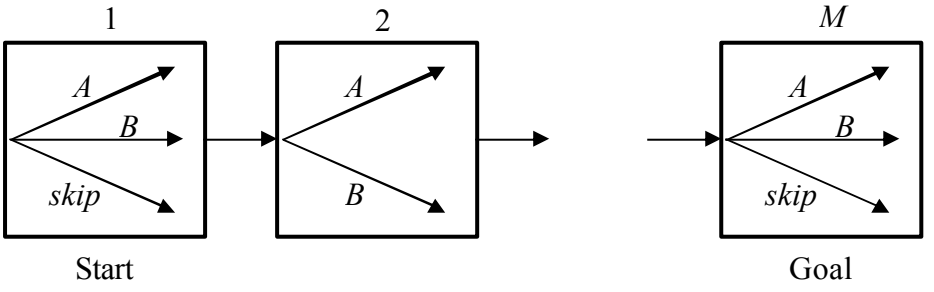


Fig. 3. An illustration of a problem instance

the skip action is added to a task with probability p and omitted with probability $1 - p$. (This probability only applies to the process of generating a random problem instance.) Action *A*, on average, uses less resources than action *B*. An agent's goal is to maximize the reward obtained during the time period. The *attractiveness* of the task can be defined as a ratio of expected reward to expected resources used to execute an action. When a sample task is generated, its expected reward is computed as follows. First, an average amount of each resource *i* used to execute an action is generated. Then the expected reward for performing the task is computed as:

$$ER = \sum_{i=1}^{n} k_i \times E[Res\_used_i] \ . \tag{1}$$

where,

k_i is a random number in the range $[0.1, 1]$
n is the number of resources

Each state is defined by the remaining resources, $R_i = \{0,\ 1, \ldots, N_i\}$, and by the current task number, $I = \{1,\ 2, \ldots, M\}$. Formally, $S = \{R_1, \ldots, R_n;\ I\}$. The start state is $S_0 = \{N_1, \ldots, N_n;\ 1\}$. An example of a terminal state is $G = \{0, \ldots, 0;\ I\}$.

4.2 Original and Abstract State Space Construction

State spaces are represented as AND/OR graphs with all unique tree nodes stored in hash tables. To construct the original and abstract state space, first, the average resource use and the reward for action one and two are precomputed. Then whether the task can be skipped is determined according to probability p. Construction of the original tree starts with the root. The root node is assigned the maximum values for each resource and variable value of 1. After that the whole graph is generated as follows. The number of successors for the first two actions is determined at random from the range [5, 15]. In case of multiple resources, it is assumed that resources are correlated, i.e. if an action requires the minimum amount of resource 1, it will also require the minimum amount of all other resources. Under this assumption, the resource values for successor states are determined as follows. The first successor always gets assigned the resource values

of the previous resource levels less some predetermined minimum resource use. The middle successor gets assigned the resource values of the previous resource levels less the precomputed average resource use. The resource levels for the rest of the successors are linearly projected from these two states. The variable value for each of the successors is determined as a previous variable value plus 1. Successor states are assigned probability values according to the linear probability model. The middle state which is associated with the average resources use has the highest probability of occurrence. The first and last states which are associated with the lowest and highest resources use have the smallest probability of occurrence. The probability values associated with all other states are linearly interpolated.

An abstract state space is constructed using the same data and assumptions as the original state space. First, the abstract root node is constructed. For example, if the initial resource is 50 and the abstraction step is 20, the initial level of resource is rounded up and the abstract root state gets assigned the resource value of 60. For the purpose of construction of abstract successors the number of successors in the original state space is assumed to be the maximum. The resource value for each successor is generated in the same way as in the original space. Then, each resource is rounded up according to the abstraction step and identical states are grouped. The probability value for each abstract successor is assigned according to the linear probability model.

4.3 Heuristic Construction and Its Application to the LAO* Algorithm

Once an abstract space is constructed, the next step is to perform a "blind" search of this abstract space. During this process, all reachable abstract states get expanded and assigned a value. These values are then used as heuristic estimates for the states in the original state space. Every time an algorithm needs to estimate a heuristic value for a state in the original space, it creates an abstract state that corresponds to the original state by rounding the resources up according to the abstraction steps and looks up the abstract state value in the hash table. The abstract state space constructed using the procedure outlined in the previous section is not guaranteed to give the exact representation of the original state space. As a result, some of the original states might not have a counterpart in the abstract state space. In case the corresponding abstract state cannot be found in the hash table, the value of the state with the same variable but higher level of the resource is used. This way the heuristic value is always overestimated and it remains admissible.

The procedure for constructing the heuristic by abstraction can be taken one step further by using multiple abstract spaces, i.e. by creating an abstraction hierarchy. The first abstract space is created in exactly the same way as before. The next abstract space is created by grouping states of the previous abstract space. This process is repeated until the top abstract space becomes trivial. The algorithm starts by performing a complete "blind" search of the top abstract level. After that at each iteration of LAO* whenever it is necessary to estimate the value of the heuristic, the next higher level of abstraction is searched. The search of the abstract space starts with the abstract state that corresponds to the state in the level below. When the second to last abstract level is searched, the heuristic estimates are simply looked up in the hash table for the top abstract level. Once an abstract space is searched at least once, the values are known for all states that

belong to the solution graph. These states can be cashed and their values can be used as heuristic estimates for the lower level in all subsequent searches.

It is worth noting that some difficulties could arise with this approach. Since abstract states are created by rounding the resources up, there could be no change in the level of resources after executing a series of actions. Therefore, in certain situations an agent can come back to the same state. In general, this is not a problem since LAO* can easily handle solutions with loops. However, it becomes a problem if an agent comes back to the same state with probability 1 because it leads to an infinite loop. Fortunately, this difficulty does not arise in the problem considered here because each state is defined by the amount of resources left and the number of the task to be performed. Even if no resources have been used while executing a task, the task number will increase. Therefore, an agent cannot come back to the same state once an action is executed. Although this problem does not contain loops and therefore could be handled by AO* algorithm, the procedure described here is general enough to handle problems with loops.

5 Experimental Results

5.1 One Resource, One Level of Abstraction

This section analyzes problems with one resource. The heuristic estimates are based on one level of abstraction. Fig. 4 shows the average number of states expanded at the base and abstract levels over 20 problems with 15 tasks and the starting level of resource of 200 units. The first bar corresponds to the average number of states expanded by "blind" search. On average, the use of abstraction step 5 is not beneficial since the algorithm expands more states than the "blind" search. All other abstraction steps can be considered beneficial since they result in some savings as compared to the "blind" search. The smallest number of states gets expanded when the abstraction step of 40 or above is used.

Fig. 5 shows the average amount of work as compared to the "blind" search (the chart on the left) and the average savings (or loss) that occur due to the use of abstraction (the chart on the right). The probability of "skip" action is 1, i.e. each task can be performed by taking an action A or an action B or the task can be skipped. The three lines on the charts on the left represent the average amount of work over 20 problems with one standard deviation band around it. The average amount of work is determined as a ratio of the total number of states expanded at both base and abstract levels over the total number of states expanded by "blind" search. The left chart shows that as the abstraction step increases, the average amount of work decreases. Similarly, the chart on the right shows that the average savings due to the use of heuristic constructed by abstraction go up as the abstraction step size increases. The maximum savings achieved are 18.1%.

The size of the problem determined by the number of unique states in a hash table can be increased by either increasing the starting level of resource or by increasing the number of tasks the agent needs to perform or by increasing both. Fig. 6 explores the relationship between the average savings due to abstraction and the size of the problem when the size of the problem increases due to increase in the starting level of resource.

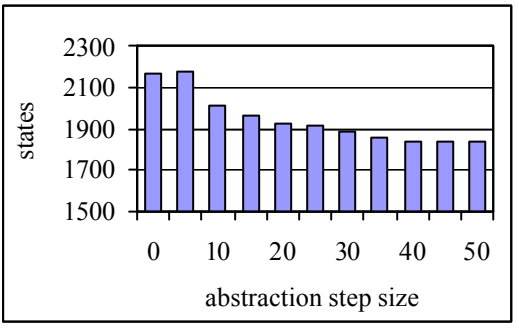


Fig. 4. Average number of states expanded both at the base and abstract level as a function of the abstraction step

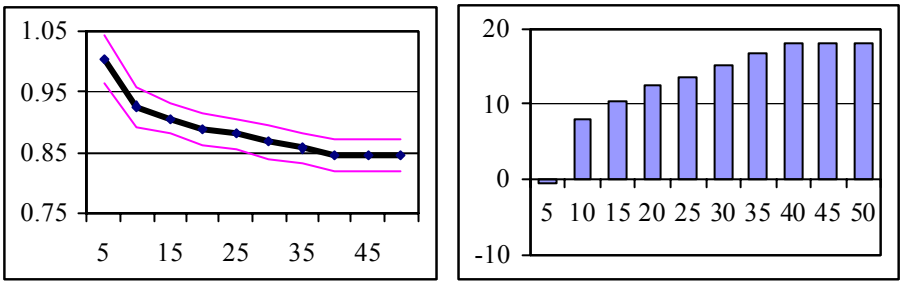


Fig. 5. Left: average amount of work as compared to blind search as a function of the abstraction step. Right: average savings in % as compared to blind search as a function of the abstraction step. An average over 20 problems with initial level of resource of 200, 15 tasks, and probability of skip action of 1

The chart on the right shows the average number of states in a hash table for each level of resource. The hash table keeps growing until initial level of resource reaches 350. After that the increase in the starting level of resource does not result in additional states being added to the hash table. When initial resource level is set at 50, it is enough to perform only a few tasks. In this case, the algorithm tries to determine which tasks should be skipped and which tasks should be performed. On the other hand, when initial level of resource is set at 500, the resource is plentiful to perform all tasks, so at each step it will be necessary to determine whether an action *A* or *B* should be preferred since the skip action will always be suboptimal (because of zero reward). The chart on the left shows the average savings due to abstraction as a function of the initial level of resource. When the level of resource is low, the savings from abstraction are the lowest (less than 15%). The savings are the highest (above 20%) when resource is abundant. A lot of the savings will occur because brunches corresponding to the skip action are suboptimal and therefore will be ignored in a heuristic search but expanded in a "blind" search. In general, the most savings occur when the problem tree has a lot of clearly

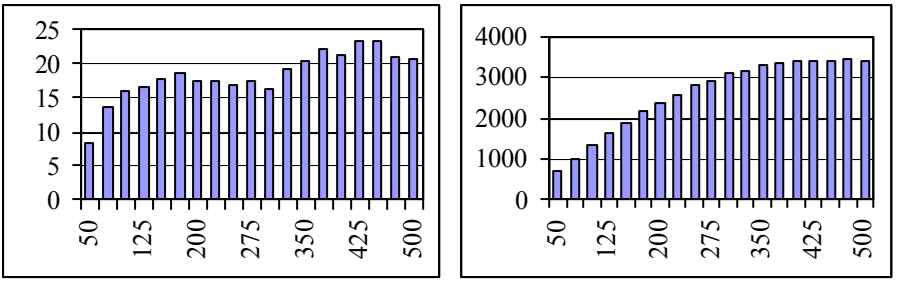


Fig. 6. Left: average savings in % as compared to blind search as a function of the initial level of resource. Right: average size of the hash table as a function of the initial level of resource. Averages are over 50 problems with 15 tasks, and probability of skip action of 1

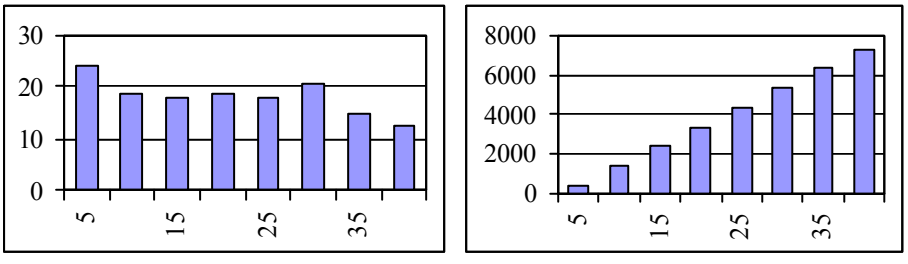


Fig. 7. Left: average savings in % as compared to blind search as a function of the number of tasks. Right: average size of the hash table as a function of the number of tasks. Averages are over 50 problems with initial level of resource set at 200, and probability of skip action of 1

suboptimal branches. Heuristics constructed by abstraction will easily identify those branches and save a lot of work at the base level.

Fig. 7 explores the relationship between the average savings due to abstraction and the size of the problem when the size of the problem increases due to increase in the number of tasks an agent has to perform. The chart on the right shows the average number of states in a hash table as a function of the number of tasks in a problem. As the number of tasks to be performed increases, so does the size of the hash table. On average, addition of 5 tasks to the problem adds roughly 1000 states to the hash table. The chart on the left shows the average savings due to abstraction as a function of the number of tasks to be performed. As in Fig. 6, the most savings occur when the initial level of resource is high relative to the number of tasks to be performed. As the number of tasks to be performed goes up, the average savings go down. The largest savings of 24.2% occur when there are only 5 tasks to be performed. In this case there is enough of the resource to perform all tasks. Savings occur largely due to the possibility to ignore those branches that correspond to the skip action while performing the heuristic search.

5.2 One Resource, Two Levels of Abstraction

In this section, an identical set of 20 problems with 15 tasks, initial level of resource of 200 and probability of skip action of 1 was solved by LAO*, first, using heuristic constructed with one level of abstraction, second, using heuristic constructed with two levels of abstraction. Fig. 8 shows graphs for one level of abstraction. Fig. 9 shows graphs for two levels of abstraction. In this case X axis values correspond to the abstraction step at the first level, s1. Abstraction step at the second level was assumed to be $s_2 = 2 \times s_1$. Comparison of the two figures shows that heuristic constructed by abstraction with one level of abstraction produces higher savings as compared to heuristic constructed by abstraction with two levels. For example, the highest possible savings with one level of abstraction are 18.1%, whereas the highest possible savings with two levels of abstraction are 17.1%.

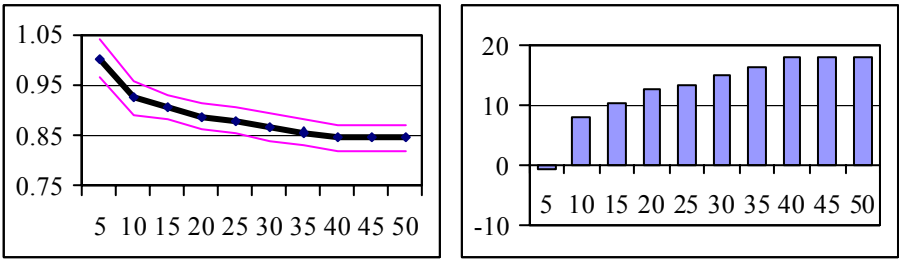


Fig. 8. One level of abstraction. Left: average amount of work as compared to blind search as a function of the abstraction step. Right: average savings in % as compared to blind search as a function of the abstraction step. An average over 20 problems with initial level of resource of 200, 15 tasks, and probability of skip action of 1

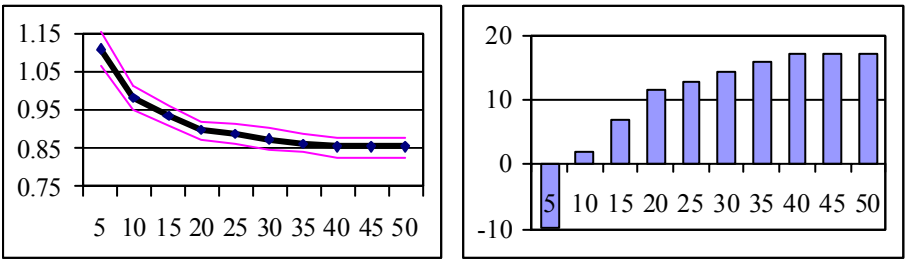


Fig. 9. Two levels of abstraction. Left: average amount of work as compared to blind search as a function of the abstraction step. Right: average savings in % as compared to blind search as a function of the abstraction step. An average over 20 problems with initial level of resource of 200, 15 tasks, and probability of skip action of 1

5.3 Two Resources, One Level of Abstraction

This section analyzes problems with two resources. The heuristic estimates are based on one level of abstraction. Fig. 10 shows the total number of states expanded at both base and abstract levels. The first bar corresponds to the average number of states expanded by "blind" search (about 3500). On average, the use of any abstraction step is beneficial. Unlike in the case with one resource, the smallest number of states (around 1960) gets expanded when the abstraction step of 10 is used.

Fig. 11 shows the average amount of work as compared to the "blind" search (the chart on the left) and the average savings that occur due to the use of abstraction (the chart on the right). On average, savings that occur due to abstraction are significantly higher than in the case with one resource for any abstraction step. The highest savings occur when abstraction step of 10 is used. In contrast, in the case of one resource the highest savings occur when the abstraction step of 40 is used.

6 Conclusions

We have shown that admissible heuristics can be generated using abstraction in stochastic domains. The results are very similar to those obtained by Holte *et al.* in deterministic settings [8]. In general, the use of abstraction in both stochastic and deterministic settings is beneficial. The actual amount of savings depends on the abstraction step. Holte *et al.* have found the abstraction radius to be large as compared to the size of the search space. As a result, only one level of abstraction is necessary. A similar conclusion can be made for the type of problems considered here. The experiments with two levels of abstraction and one resource show that the use of one level of abstraction results in higher savings as compared to two levels. In case of one level of abstraction and one resource, an abstraction step of 40 turns out to be the most beneficial; in case of two resources, an abstraction step of 10 is the most beneficial. In general, the amount of savings depends on the difficulty of the problem. Problems with two resources result

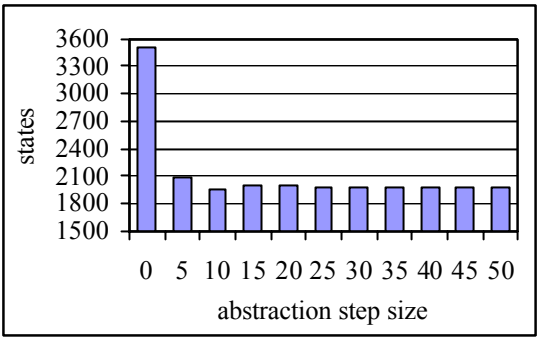


Fig. 10. Average number of states expanded both at the base and abstract level as a function of the abstraction step

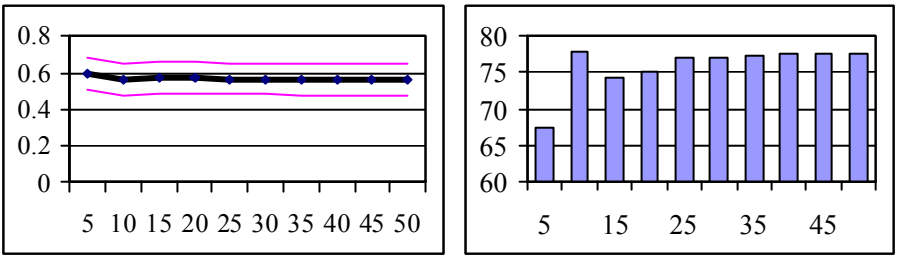


Fig. 11. Left: average amount of work as compared to blind search as a function of the abstraction step. Right: average savings in % as compared to blind search as a function of the abstraction step. An average over 20 problems with initial level of resource 1 of 50, resource 2 of 50, 7 tasks, and probability of skip action of 1

in much higher savings as compared to the problems with one resource. We expect the savings to grow with the number of resources.

One benefit of our approach is that it is designed to avoid visiting the entire state space either during search or any preprocessing stage. Although the abstract space is created in advance, it is generated independently of the original space. Moreover, there is no need to go through the entire base-level state space in order to search the abstract space. Another source of savings is the fact that the search in the abstract space is only through reachable states, not all states.

Because it is generally much harder and less intuitive to design admissible heuristics for stochastic domains, it is beneficial to design automated techniques based on abstraction such as the one we present in this paper. Moreover, because it is relatively easy to transform LAO* into an approximation anytime algorithm, the result of this work facilitate the development of both exact and approximate algorithms for search in stochastic domains.

Acknowledgments

Support for this work was provided in part by the National Science Foundation under grant number IIS-0328601.

References

1. Barto, A.G., Mahadevan, S.: Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Dynamic Systems: Theory and Applications* **13** (2003) 41–77
2. Boutilier, C., Dearden, R.: Approximating Value Trees in Structured Dynamic Programming. In *Proc. of the Thirteenth International Conference on Machine Learning* (1996)
3. Culberson, J.C., Schaeffer, J.: Pattern Databases. *Computational Intelligence* **14(3)** (1998) 318–334
4. Dean, T., Pack Kaelbling, L., Kirman, J., Nicholson, A.: Planning with Deadlines in Stochastic Domains. In *Proc. of the Eleventh National Conference on Artificial Intelligence* (1993) 574–579

5. Dearden, R., Boutilier, C.: Abstraction and Approximate Decision-Theoretic Planning. *Artificial Intelligence* **89** (1997) 219–283
6. Hansen, E.A., Zilberstein, S.: LAO*: A Heuristic Search Algorithm that Finds Solutions with Loops. *Artificial Intelligence* **129** (1-2) (2001) 35–62
7. Holte, R.C., Drummond, C., Perez, M.B., Zimmer, R.M., MacDonald, A.J.: Searching with Abstractions: A Unifying Framework and New High-Performance Algorithm. In *Proc. of the Canadian Artificial Intelligence Conference* (1994) 263–270
8. Holte, R.C., Perez, M.B., Zimmer, R.M., MacDonald, A.J.: Hierarchical A*: Searching Abstraction Hierarchies Efficiently. In *Proc. of the Thirteenth National Conference on Artificial Intelligence* (1996) 530–535
9. Korf, R.E., Felner, A.: Disjoint Pattern Database Heuristics. *Artificial Intelligence* **134(1-2)** (2002) 9–22
10. Korf, R.E.: Finding Optimal Solutions to Rubik's Cube Using Pattern Databases. In *Proc. of the Fourteenth National Conference on Artificial Intelligence* (1997) 700–705
11. Pearl, J.: *Heuristics*. Addison-Wesley (1984)
12. Preditis, A.: Machine Discovery of Admissible Heuristics. *Machine Learning* **12** (1995) 165–175
13. St-Aubin, R., Hoey, J., Boutilier, C.: APRICODD: Approximate Policy Construction Using Decision Diagrams. *Neural Information Processing Systems* **13** (2000)
14. Tash, J., Russell, S.: Control Strategies for a Stochastic Planner. In *Proc. of the Twelfth National Conference on Artificial Intelligence* (1994) 1079–1085
15. Valtorta, M.: A New Result on the Computational Complexity of Heuristic Estimates for the A* Algorithm. *Artificial Intelligence* **55(1)** (1992) 129–142

Synthesizing Plans for Multiple Domains

Abdelbaki Bouguerra and Lars Karlsson*

Applied Autonomous Sensor Systems Center,
Örebro University, Sweden
{abdelbaki.bouguerra, lars.karlsson}@tech.oru.se

Abstract. Intelligent agents acting in real world environments need to synthesize their course of action based on multiple sources of knowledge. They also need to generate plans that smoothly integrate actions from different domains. In this paper we present a generic approach to synthesize plans for solving planning problems involving multiple domains. The proposed approach performs search hierarchically by starting planning in one domain and considering subgoals related to the other domains as abstract tasks to be planned for later when their respective domains are considered. To plan in each domain, a domain-dependent planner can be used, making it possible to integrate different planners, possibly with different specializations. We outline the algorithm, and the assumptions underlying its functionality. We also demonstrate through a detailed example, how the proposed framework compares to planning in one global domain.

1 Introduction

A considerable amount of work in AI planning focuses on the use of abstraction to reduce the search space, where planning takes place at successive levels of more details. Hierarchical planning is such a paradigm that relies on abstraction and goal decoupling to produce effective plans [14],[17],[12],[5]. The planning problem is specified as a set of abstract tasks to achieve with ordering constraints over them. The planning process, repeatedly, refines the abstract tasks into more detailed tasks until the plan is composed only of executable tasks. Abstractions have mainly been supplied by the user as part of the knowledge bases used by the planner. However different approaches have been proposed to learn abstractions from the description of the planning problems [8],[4].

Decomposition of the planning problem into subproblems is also an approach aiming at reducing search complexity [16],[1]. The partitioning of the initial planning problem focuses on producing subproblems with minimum interaction in order to be able to find an efficient solution. It is worth noting that problem partitioning is generally combined with abstraction techniques to control the interaction between the different sub-components of the planning problem [10],[11].

* This work has been supported by the Swedish KK foundation and the Swedish research council.

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 30–43, 2005.

In this paper we propose a framework to synthesize plans to solve planning problems involving multiple domains through the use of abstraction and goal ordering. In fact, goal and subgoal ordering approaches have been demonstrated to be effective ways in solving planning problems [13]. As demonstrated in [9], a total order relation between increasing sets of goals allows incremental planning by focusing search on the goals that appear earlier, leading to improved planning performances.

To solve problems involving multiple domains, one can envisage to solve in each domain the portion of the problem related to it and then glue all the results in a global plan, but doing so might result in degraded execution of the overall plan, because of the localized reasoning. Our work on plan execution on board mobile robots has motivated us to find a general approach that can utilize reasoning over different domains using different planners for problem solving so the resulting plan would execute smoothly and efficiently. Therefore the proposed approach guarantees to find a plan in an incremental way that interleaves actions from the different domains when only it is needed.

The general idea of our approach is to act on a set of domains ordered according to which domain gets its goals achieved earlier. The planning problem is solved incrementally starting with the leftmost domain, and going all the way to the last one. If a domain D, ordered before another domain D', needs to accomplish a subgoal involving the actions of D', then D places a request in the plan for D' to accomplish the desired subgoal. At a later stage, when planning to solve in D', the algorithm can use the actions of D' to solve the request. The planner used to plan in a particular domain can be a domain-(in)dependent planner, meaning that it is possible to integrate different efficient specialized planners to solve the global planning problem.

In the next section we detail the assumptions used to find a plan in multiple domains as well as operator transformation to reflect the interaction between the involved domains. Next we give a global overview of the approach and how planning in one domain introduces ordered abstract tasks to be solved in the subsequent domains. Section 4 outlines the hierarchical algorithms used to solve the multiple-domain planning problem. Before concluding we demonstrate the performance of the approach on two domains.

2 Domains Interaction

In this section we discuss the assumptions underlying the interactions between the planning domains, as well as the extensions to be made to the syntax of planning operators in order to be able to use the proposed framework. As stated before, the approach supposes that the agent has access to a planning system employing a domain-independent planner or a set of domain-dependent planners.

Domains are defined in the usual way as consisting of a set of operators and a set of fluents, where the operators of one domain can use literals from other domains in their preconditions and effects. We use an incremental approach to plan in the different domains i.e. when trying to solve a planning problem in a

particular domain, the achieved goals, of solved problems of the other domains, have to be maintained. This incremental approach makes it possible to reduce search complexity, because goals do not have to be established, then destroyed, then established again [9]. This restriction leads us to the following consideration:

- **Consideration 1.** If an operator in a domain D_1 uses in its preconditions a literal l from another domain D_2, then finding a plan to achieve goals in D_1 would violate achieved goals in D_2, because the literal l might be sub-goaled in D_1 by the corresponding plan. Therefore the framework has to make sure that all the goals of D_1 are planned for first before planning for the goals of D_2. Note that this is coherent with the ordered monotonic refinement property [8].

Example 1. Suppose that the planning system has two domains: navigation, and blocks. The navigation domain is used to move a mobile robot equipped with an arm between rooms and corridors. The blocks domain is used to rearrange blocks in towers in different rooms. To manipulate blocks in a room r_1, the robot has to be in room r_1 too. Therefore the blocks domain operators (pick-up, put-down, stack, and unstack) use a literal in their preconditions to impose such restriction. Now, suppose that the agent wants to achieve the two goals $g_1 = (\text{on } b\ a)$, and $g_2 = (\text{robot-at} = r_2)$ (where both blocks a and b are in room r_3). If the agent achieves g_2 first then solving g_1 will violate g_2; because the robot has to move to room r_3 to be able to stack b on a. Consequently, g_1 has to be achieved before g_2.

We also want to keep the effects involving fluents from a domain D under the control of D. The aim of this restriction is to localize planning within the domains. This leads us to the second consideration:

- **Consideration 2.** If an operator o in domain D_1 achieves as a side effect a literal l from another domain D_2, then an abstract version of o is created in D_2 where only the preconditions related to D_2 are maintained, and the effects related to D_1 are posted as a task to be achieved in D_1.

Example 2. Considering the previous example. In the navigation domain, moving the robot from one room to another while holding a block will also move the block as a side effect. Therefore an abstract version of this operator is created in the blocks domain to reflect this change to the state of the block.

2.1 Domains Representation

Let $\mathscr{D} = \{D_1, D_2, \cdots, D_m\}$ be the set of the domains used by the planning system. A domain D_i is defined as a set of fluents F_i and a set of operators O_i. An operator $o \in O_i$ is a couple $\langle Pre(o), Eff(o) \rangle$, where *Pre(o)* are the preconditions of the operator, and *Eff(o)* are the effects (positive and negative) of the operator. Following the consideration 1, the preconditions *Pre(o)* component has the following syntax:

$$\begin{aligned} &\textit{Pre(o)} :: ((\textit{local} : \phi_i) \\ &\qquad (\textit{foreign} : (D_j : \psi_j)^+)^*) \end{aligned}$$

where ϕ_i is a formula written only over the (local) fluents of the domain D_i, and ψ_j is a formula written only over the fluents of domain D_j $(i \neq j)$. This syntax reflects that the preconditions of the operator o might have a local precondition expressed as a formula over the fluents F_i of the current domain, and a list of foreign preconditions defined over the fluents of the other domains.

If o is an abstract version of another operator in another domain D_j $(j \neq i)$ (consideration 2), then the effects of o have the following form:

$$\begin{aligned} &\textit{Eff(o)} :: ((\textit{local} : \textit{eff}_i) \\ &\qquad (\textit{foreign} : (D_j : \textit{eff}_j)^+)^*) \end{aligned}$$

where $\textit{eff}_i$ are effects over the fluents of the local domain D_i and $\textit{eff}_j$ are effects over the fluents of domain D_j $(j \neq i)$.

Example 3. The following is an operator from the blocks domain to pick up a block on the table at a location specified by the variable ?*loc*.

(pick-up ?b ?loc):	
param:	?b - BLOCK, ?loc - LOCATION
Pre:	((local: (and (clear ?b)(on-table ?b)(arm-free)(object-at ?b = ?loc))) (foreign: (navigation: (robot-at = ?loc)))
Eff:	((local: (and (holding ?b)(clear ?b = f)(arm-free = f) (on-table ?b = f))))

The *foreign* part of the precondition specifies that the fluent (robot-at = ?*loc*) from the navigation domain has to be satisfied before executing the operator i.e. the robot must be at the same location as the block.

Example 4. The operator (move $?l_1$ $?l_2$) from the navigation domain is used to move a robot from location $?l_1$ to another location $?l_2$. If the robot is holding a block ?*b*, then the block changes location too (i.e. (object-at $?b = ?l_1$)). So an abstract operator has to be created in the blocks domain to reflect this value change to the fluent object-at. When planning in the blocks domain, the abstract operator is used the same way as the other operators to cause state change, except that it does not appear in the plan as its real effects are achieved by the operator (move $?l_1$ $?l_2$) of the navigation domain (it is for this reason it is qualified as abstract).

(move-block ?b ?l1 ?l2): ABSTRACT	
param:	?b BLOCK, ?l1 ?l2 - LOCATION
Pre:	((local: (and (holding ?b)(object-at ?b = ?l1)))
Eff:	((local: (object-at ?b = ?l2)) (foreign: (navigation: (robot-at = ?l2))))

2.2 Ordering Domains

Consideration 1 imposes on the multi-domain plan synthesizer to find a total order on the set of domains $\mathscr{D} = \{D_1, D_2, \cdots, D_m\}$. A domain D_i is ordered before another domain D_j (noted: $D_i \prec D_j$) if at least one of the operators of D_i uses in its preconditions a fluent from D_j:

$$\exists f_j \in F_j, \exists o \in O_i : f_j \text{ appears in } Pre(o) \Rightarrow D_i \prec D_j.$$

The total order can be directly given by the user, or it can be extracted automatically as the result of a topological sort applied on a graph whose nodes represent the domains and arcs represent the constraint $\prec$.

3 Multiple-Domain Planning Overview

To be able to synthesize a plan to solve a planning problem involving the achievement of goals related to more than one domain, we need first to find a total order on the involved domains, and create abstract operators to fulfill the second requirement as described in the previous section. Once this is done, we can solve the planning problem in an incremental way: starting in the left-most domain we solve its planning problem, then the resulting plan is passed to the next domain (according to their order) where the abstract tasks related to the new domain are solved in the order they appear in the plan. This process continues until reaching the right-most domain where the plan would be completely refined.

3.1 Planning in One Domain

A planning problem in a domain D_i is specified by the initial state expressed as a conjunct of fluents from F_i, a goal state expressed as a first order logic formula, and the set of operators O_i. To solve a planning problem to achieve goals in domain D_i, the planner associated with D_i selects an instantiated operator $o \in O_i$ to insert in the plan, if its local preconditions are satisfied in the current state s defined over F_i. Since the foreign preconditions of the operator o have also to be satisfied in their respective domains, the planner prepends to o requests to achieve the foreign preconditions in their respective domains. If an operator has more than one foreign component, the planner has to make sure to order the requests according to the order of their domains. i.e. if the foreign components of the preconditions *Pre(o)* of o are $(foreign : (D_1 : \psi_1)(D_2 : \psi_2))$ such that $D_1 \prec D_2$, then as a result of selecting o, the planner inserts the following in the plan:

$$\texttt{(achieve } (D_1 : \psi_1)\texttt{)};\texttt{(achieve } (D_2 : \psi_2)\texttt{)} \ ; o$$

where $\texttt{(achieve } (D_i : \psi_i)\texttt{)}$ formulates a request to achieve the conditions ψ_i in domain D_i.

If o is an abstract operator with foreign effects i.e. $(foreign : (D_j : eff_j)) \in$ *Eff(o)*, then the planner inserts those foreign effects as requests to be achieved in their respective domains i.e. `(achieve` $(D_j : eff_j)$`)`. Note that in this case, only the foreign effects of o are inserted in the plan as abstract tasks, but not o itself.

Each request, posted on a foreign component defines an abstract task to be achieved in its domain. This means that a plan might encompass unsolved abstract tasks inserted by the planners of the antecedent domains. The unsolved tasks are further refined when planning in the subsequent domains.

Example 5. The instantiated blocks domain operator (pick-up b_1 r_1) is applicable in a state s if the local formula (and (clear b_1)(on-table b_1)(arm-free)(block-at b_1= r_1)) holds in s. If it is selected by the planner then, the planner prepends it with the abstract task "`(achieve` (Navigation: (robot-at = r_1)`)`" which is solved later when planning in the navigation domain.

3.2 Planning in Multiple Domains

Figure 1 gives an overview of how to synthesize a plan involving three domains $\mathscr{D} = \{D_1, D_2, D_3\}$ such that $D_1 \prec D_2 \prec D_3$. The goals involving one domain are considered as an abstract task to achieve. Therefore, a task is created on the final goals of each domain, giving us three initial abstract tasks: $T_{init}(D_1), T_{init}(D_2), T_{init}(D_3)$ where each $T_{init}(D_j)_{j=1,2,3}$ is formulated as `(achieve` $(D_j$: goals of $D_j)$`)`. The initial version of the global abstract plan is created by ordering the three initial abstract tasks according to the order defined over their respective domains i.e. $T_{init}(D_1) \prec T_{init}(D_2) \prec T_{init}(D_3)$.

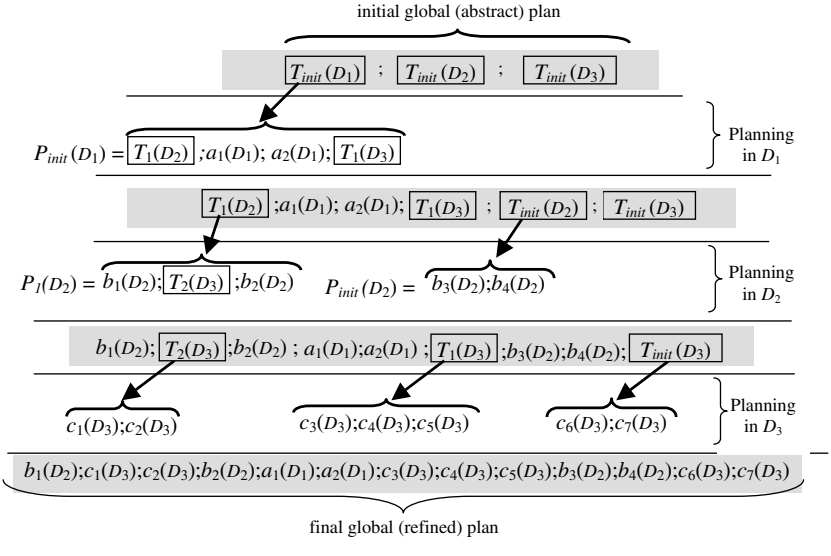


Fig. 1. Global view of planning in three domains

The plan synthesizing process proceeds top-down. It starts by planning in D_1 to solve $T_{init}(D_1)$ which yields a plan $P_{init}(D_1) = T_1(D_2); a_1(D1); a_2(D1); T_1(D_3)$ that contains two abstract tasks $T_1(D_2)$(resp. $T_1(D_3)$) to be achieved in D_2 (resp. D_3) and two instantiated operators of domain D_1: $a_1(D1)$ and $a_2(D1)$. $T_{init}(D_1)$ is then replaced by $P_{init}(D_1)$ to produce a global plan which is subsequently refined in D_2. When planning in D_2, the planner plans to solve the abstract tasks related to D_2 in the same order they appear i.e. it plans to solve $T_1(D_2)$ before planning to solve $T_{init}(D_2)$. The next step replaces the abstract tasks just planned for by their corresponding plans ($T_1(D_2)$ is replaced by $P_1(D_2)$, and $T_{init}(D_2)$ is replaced by $P_{init}(D_2)$). The resulting plan is next refined in the last domain D_3 where abstract tasks related to D_3 are solved and replaced by plans that contain only instantiated operators from D_3. At this stage the resulting plan is composed only of instantiated operators and it solves all the goals related to the three domains. Please note that there might be backtracking to the previous domain or just within a domain itself if a task can not be solved.

4 The Planning Algorithm

The multi-domain planning algorithm shown in Fig. 2. is a forward chaining algorithm. It takes as input a set of initial states S_0, a set of goals to achieve G, and a list of all the domains *Domains* given in the order defined in section 2. The elements of S_0 are the initial states of the domains involved in planning. G comprises goal sets, each of which is related to one domain in *Domains*.

The algorithm builds for every domain $D_i \in Domains$ a task which is simply expressed as "`(achieve` $(D_i : g_{D_i})$`)`", where $g_{D_i} \in G$ is the goal set related to domain D_i. An abstract version of the global plan *GlobalP* is created by ordering the tasks of the different domains according to the order of their respective domains (the Init phase). After the initialization phase, the algorithm retrieves the first domain D from *Domains* (step 2), and extracts all the abstract tasks D_{Tasks} from *GlobalP* related to D keeping them in the same order as they appear in *GlobalP* (step 4). The procedure "Find-Plan" is called to compute a plan D_{plan} to solve the ordered list of abstract tasks D_{Tasks} in D (step 6). D_{plan} is actually a list of sub-plans each solving one task in D_{Tasks}. If all the tasks in D_{Tasks} are solvable in D (i.e $D_{plan} \neq fail$), then *GlobalP* is refined by substituting every abstract task that appears in D_{Tasks} by the portion of the sub-plan that solves it (step 8) (this is a refinement step that iterates over the elements of *GlobalP* to replace an abstract task related to D by a sub-plan from D_{plan} if it solves it).

The same process repeats with the rest of the domains until finishing all of them. When all the domains are planned in, the global plan *GlobalP* is completely refined. *GlobalP* is returned as a solution for the multiple-domain planning problem (step 1). In case, there is no plan to solve the tasks in D_{Tasks} (step 7), The multiple-domain planner returns *fail* (step 13).

Since we might have backtracking when planning in the subsequent domains (steps 9 and 10), "next(Find-Plan($s_{0d} \in S_0, D_{Tasks}, D$))" (step 6) is supposed

Input:
$S_0 = \{s_{0i}$: initial state of domain $D_i\}$
$G = \{g_{Di}$: goal set of domain $D_i\}$
Domains: the ordered list of domains to use
Output:
A plan that achieves G starting from S_0
Init:
$GlobalP$ = build-and-order-abstract-tasks(G)

Algorithm MD-Plan($S_0, GlobalP, Domains$)
1. **if** *Domains* is empty then return $GlobalP$ **endif**
2. D = first of *Domains*
3. RD = rest of *Domains*
4. D_{Tasks} = extract-tasks($GlobalP, D$)
5. **do**
6. D_{plan} = next(Find-Plan($s_{0d} \in S_0, D_{Tasks}, D$))
7. **if** $D_{plan} \neq fail$ **then**
8. $GlobalP$ = substitute-plan($GlobalP, D_{plan}$)
9. RD_{plan} = MD-Plan($S_0, GlobalP, RD$)
10. **if** $RD_{plan} \neq fail$ return RD_{plan} **endif**
11. **endif**
12. **until** $D_{plan} = fail$
13. return $fail$

END

Fig. 2. The multi-domain planning algorithm

to give the next valid plan D_{plan} solving the abstract tasks D_{Tasks} in domain D i.e. a plan that has not been considered yet (the call to the function "next" can be considered as iterating over a set of valid plans that solve D_{Tasks} in D).

The algorithm used to solve an ordered list of tasks in one domain is outlined in Fig. 3. It gets as input the initial state related to the domain s_0, an ordered list of tasks to solve $Tasks$, and the relevant domain D. The algorithm retrieves the first task g from $Tasks$ (step 2) and calls a planner to solve it starting in the initial state s_0 (step 5). If the task is solvable, the planner returns a plan P_g that solves it along with the goal state s_g where the task g is satisfied. As mentioned before, the planner can be specialized to solve planning problems related to the current domain, or a generic planner (domain-independent). In the next recursive call (step 7), the algorithm tries to solve the rest of the tasks starting from s_g this time i.e. the initial state for the next task is s_g. The algorithm continues recursively doing so until solving all the tasks where it returns *success* (step 1). If *success* is returned then the plan that solves all the ordered tasks specified in $Tasks$ is the concatenation of the plans solving each task apart (step 9). Please note that as in the previous algorithm, the use of next (in step 5) returns the next valid pair (plan, goal-state). This means that next iterates over a set of

Input:
s_0 : initial state
$Tasks$: a list of ordered tasks to achieve
D : domain to use
Output:
A plan that achieves $Tasks$ starting from s_0

Algorithm Find-Plan$(s_0, Tasks, D)$
1. **if** $Tasks$ is empty **then return** *success* **endif**
2. g = the first task in $Tasks$
3. R = the rest of $Tasks$
4. **do**
5. (P_g, s_g) = next(PLANNER(s_0, g, D))
6. **if** $P_g \neq fail$ **then**
7. P_R = Find-Plan(s_g, R, D)
8. **if** $P_R \neq fail$ **then**
9. **return** $P_g; P_R$
10. **endif**
11. **endif**
12. **until** $P_g = fail$
13. **return** *fail*

END

Fig. 3. The Find-Plan planning algorithm

plans that can be generated by the planner to solve one planning problem. A failure is returned (step 12) if the current task can not be solved, or if all the plans that solve the current task make the subsequent tasks unsolvable.

5 Detailed Example

This section demonstrates the performance of the proposed approach in the two domains navigation and blocks, shown in Fig. 4. A planning problem in the navigation domain consists in moving a mobile robot, equipped with an arm, from one location to another. The different locations are connected by doors that can be open or closed. The blocks domain is the standard AI planning bench mark domain used to form towers of blocks according to a set of constraints over the positions of the blocks. In our experiment, it is the mobile robot that is responsible of forming the towers of blocks. Furthermore, a block is constrained to be at a specific location. Consequently, in order to execute an action in the blocks domain, the mobile robot has to be at the same location as the relevant blocks (the blocks that undergo the action). As a result of consideration 1, the blocks domain is augmented with the abstract operator "move-block" derived from the navigation domain operator "move-in". In this scenario the blocks

Domain: Blocks

(pickup ?b ?l)
param: `?b - BLOCK, ?l - LOCATION`
Pre: `((local: (and (clear ?b)(on-table ?b) (arm-free) (object-at ?b = ?l))) (foreign: (navigation: (robot-at = ?l)) ))`
Eff: `((local:(and (holding ?b)(clear ?b = f) (on-table ?b = f)(arm-free = f))))`

(putdown ?b ?l)
param: `?b - BLOCK, ?l - LOCATION`
Pre: `((local: (and (holding ?b)(object-at ?b = ?l))) (foreign: (navigation: (robot-at = ?l) )))`
Eff: `((local:(and (holding ?b = f)(clear ?b = t) (on-table ?b = t)(arm-free = t))))`

(unstack ?a ?b ?l)
param: `?a ?b - BLOCK, ?l - LOCATION`
Pre: `((local: (and (clear ?a)(on ?a ?b)(object-at ?b = ?l) (object-at ?a = ?l) (arm-free))) (foreign: (navigation: (robot-at = ?l) )))`
Eff: `((local:(and (holding ?a)(clear ?b)(clear ?a = f) (on ?a ?b = f)(arm-free = f)))))`

(stack ?a ?b ?l)
param: `?a ?b - BLOCK, ?l - LOCATION`
Pre: `((local: (and (holding ?a)(clear ?b) (object-at ?b = ?l)(object-at ?a = ?l))) (foreign: (navigation: (robot-at = ?l))))`
Eff: `((local: (and (holding ?a = f)(clear ?b = f) (clear ?a) (on ?a ?b)(arm-free))))`

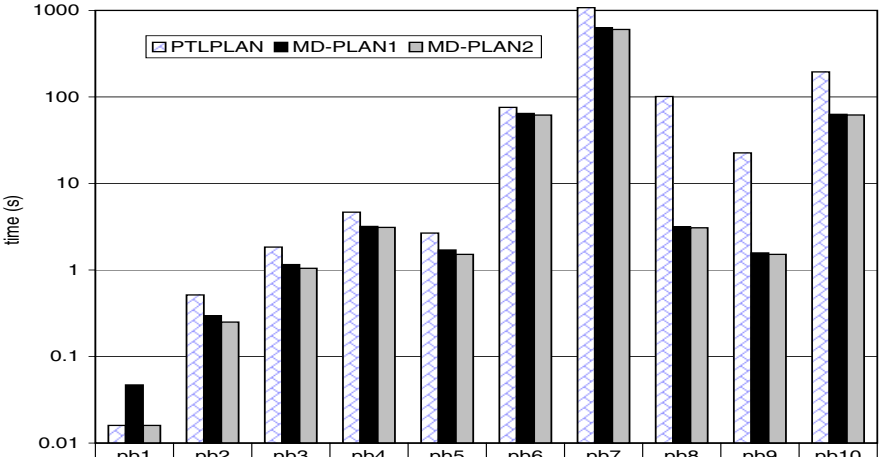
(move-block ?b ?l1 ?l2): ABSTRACT
param: `?b BLOCK, ?l1 ?l2 - LOCATION`
Pre: `((local: (and (holding ?b)(object-at ?b = ?l1)))`
Eff: `((local: (object-at ?b = ?l2)) (foreign: (navigation: (robot-at = ?l2))))`

Domain: Navigation

(move-in ?l ?2)
param: `?l1 ?l2 - LOCATION`
Pre: `((local: (and (robot-at = ?l1) (exists (?d - DOOR)(and (part-of ?d ?l1) (part-of ?d ?l2)(closed ?d = f))))))`
Eff: `((local: (robot-at = ?l2)))`

(open-door ?d)
param: `?d - DOOR`
Pre: `((local: (and (closed ?d)(exists (?l -LOCATION) (and (part-of ?d ?l)(robot-at = ?l))))))`
Eff: `((local:(closed ?d = F))`

Fig. 4. The blocks and navigation domains



	pb1	pb2	pb3	pb4	pb5	pb6	pb7	pb8	pb9	pb10
PTLPLAN	0.016	0.515	1.843	4.656	2.672	75.812	1075.3	101.2	22.563	194.8
MD-PLAN1	0.047	0.296	1.156	3.188	1.703	64.547	632.11	3.172	1.578	63.031
MD-PLAN2	0.016	0.25	1.047	3.109	1.515	61.765	602.06	3.078	1.515	61.844

Fig. 5. Execution times for navigation and blocks domains

domain is ordered before the navigation domain as a result of considerations 1 & 2 stated in section 2.

To evaluate the generation of plans to solve planning problems involving both domains (blocks and navigation), we compared execution times taken by our approach against the execution times taken by the domain-independent planner PTLPLAN [6]. In order to use PTLPLAN, both domains were collapsed in one global domain. We used two versions of the multiple domain planning approach. In the first version called MD-PLAN1, the planner used to plan in the navigation domain as well as the blocks domain is PTLPLAN. In the second version, called MD-PLAN2, we used a specialized planner to plan in the navigation domain, and PTLPLAN to plan in the blocks domain. The navigation specialized planner is a graph-based search algorithm that returns all the different plans that can lead from one location to another one.

The tests were run on 10 problems, with different numbers of blocks and rooms. The different problems involved forming towers of blocks in different rooms which involved moving the blocks from their initial location to their goal location. Figure 5 shows a bar chart diagram (where the values of the y axis are logarithmic) as well as a table of the executions times in seconds taken by PTLPLAN, MD-PLAN1, and MD-PLAN2 to solve the ten problems.

The diagram shows that the two versions of the proposed approach outperform PTLPLAN applied to the two domains as one global domain. It is also worth noting that the second version MD-PLAN2 is slightly faster than the first version MD-PLAN1, which is clearly an advantage of using specialized domain-dependent planners. The performance of the proposed approach against planning in one global domain can be attributed to localized planning in the respective

domains and solving interactions through the use of abstract tasks and their orderings. We believe that many real world scenarios involve domains that have little interaction such as the blocks and navigation domains, therefore defining an ordered structure over them and solving problems local to each domain within the domain itself would greatly reduce the complexity of search.

6 Related Work

In this section we review some of the systems close to the proposed approach. Alpine [7, 8] is one of the first systems proposed to automate the generation of abstraction hierarchies for a specific planning problem by grouping literals appearing in the preconditions and effects of operators according to predefined constraints over operators conflicts. The partitions are then topologically ordered forming a directed graph of abstraction levels. Alpine solves the planning problem in the simpler abstract space, then refines the abstract solution at successive levels of detail by inserting operators to achieve the conditions that were ignored in the higher levels of abstraction. Alpine relies on the "Ordered Monotonicity Property" to refine abstract plans: the refinements of the abstract plans maintain the literals established in the higher levels of abstraction i.e. make sure not to violate what it has been achieved at higher levels. But as mentioned in [15] Alpine does not guarantee the construction of good hierarchies, because it ignores variable binding conflicts. Our approach differs from planning with alpine in different ways. First, Alpine plans with abstract spaces specific to planning problems, ours on the other hand uses abstraction on the domain level. Second, in Alpine each intermediate state in an abstract plan forms an intermediate goal (task to achieve) at the next level of detail; in our approach an abstract plan can have tasks to be achieved at all the next levels of detail.

Collage [10] partitions the overall planning problem into regions of actions and constraints over them. The localized partitioning relies on building a DAG of abstract partitions from constraints over actions and their scope i.e. their relevance to actions of the plan associated with their region and its subregions. The planning algorithm has to maintain the consistency between the different search regions involving a considerable amount of jumping between them to cope with their interactions.

STRPLAN [11] is also a planning system that decomposes the original planning problem into sub-regions. The system uses a language that allows it to specify domain sub-regions and specify local planners to them. To find a global plan, a centralized control module coordinates the local planners by solving constraints over their sub-regions plans.

Perhaps the most related approach to ours is the one reported in [1] where the planning domain is partitioned into sub-domains and organized in a tree structure. The planning process works in two stages: first, abstracted actions coded as complex messages are computed at each sub-domain. Starting with the leaf sub-domains, the abstract action are added to the parent sub-domain until reaching the root sub-domain which contains the global planning problem

goal. The root sub-domain uses all its actions and the abstract actions from its descendant nodes to find an abstract plan that solves the original planning problem. The second stage consists in refining the plan found at the root node by replacing all the abstract actions in the root plan by a sub-plan from the actions of the sub-domains. The main difference with our approach is the construction of the abstract actions at every sub-domain to represent all plans the sub-domain can find to affect the fluents it shares with its parents. In our approach, only one plan is constructed at a domain level, this plan solves the goals of the corresponding domain and introduces new order constraints on the subgoals to be solved in the subsequent domains.

The proposed approach is also comparable to Hierarchical Task Networks "HTN" Planners such as SHOP [2], SHOP2 [12] and UMCP [3]. However HTN planning relies on an expert to hand-code the procedures that are used to refine abstract tasks which is error-prone and not easy for certain domains. Our approach relies on first principles planning to build abstract tasks automatically during planning.

7 Conclusion

We have presented in this paper an approach to synthesize plans involving multiple domains. The approach assumes that a total order of the domains exists to perform plan synthesizing by hierarchically refining abstract tasks defined as subgoals in their respective domains. The main advantages of using such approach, besides search complexity reduction, are its simplicity, and the possibility to use specialized planners when planning in the different domains. It can also be seen as an alternative to using one big global planning domain making it possible to write sub-domains by different experts. The use of abstraction to solve subgoals seems to be a natural way that humans use in their daily life when performing tasks implying different fields of knowledge. The assumption of a total order between domains limits the applicability of the proposed approach to domains with the kind of interactions discussed in section 2. In our future work, we will investigate how to extend the planning algorithms to domains that can not be totally ordered, i.e when there are cycles between the domains in terms of the relation $\prec$. We also envisage to integrate the presented framework with the plan executor implemented on board our mobile robots.

References

1. E. Amir and B. Engelhardt. Factored planning. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence(IJCAI)*, pages 929–935, Acapulco, Mexico, 2003. Morgan Kaufmann.
2. A. Lotem D. Nau, Y. Cao and H. Muñoz-Avila. Shop: Simple hierarchical ordered planner. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 968–973, 1999.

3. K. Erol, J. A. Hendler, and D. S. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In *Artificial Intelligence Planning Systems*, pages 249–254, 1994.
4. E. Fink and Q. Yang. Automatically abstracting the effects of operators. In James Hendler, editor, *Proceedings of the 1st International Conference on Artificial Intelligence Planning Systems (AIPS92)*, pages 243–251, College Park, Maryland, USA, 1992. Morgan Kaufmann.
5. F. Giunchiglia, A. Villafiorita, and T. Walsh. Theories of abstraction. *Artificial Intelligence Communications*, 10(3-4):167–176, 1997.
6. L. Karlsson. Conditional progressive planning under uncertainty. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 431–438, 2001.
7. C. A. Knoblock. Learning abstraction hierarchies for problem solving. In Thomas Dietterich and William Swartout, editors, *Proceedings of the 8th National Conference on Artificial Intelligence*, Menlo Park, California, 1990. AAAI Press.
8. C. A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68(2):243 – 302, 1994.
9. J. Koehler. Solving complex planning tasks through extraction of subproblems. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS98)*, pages 62–69, 1998.
10. A. L. Lansky and L. Getoor. Scope and abstraction: Two criteria for localized planning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1612–1619, 1995.
11. R. B. Llavori. Strplan: A distributed planner for object-centered application domains. *Applied Intelligence*, 10(2-3):259–275, 1999.
12. D. S. Nau, T. C. Au, O. Ilghami, U. Kuter, W. Murdock, D. Wu, and F. Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20:379–404, December 2003.
13. J. Porteous, L. Sebastia, and J. Hoffmann. On the extraction, ordering, and usage of landmarks in planning. In *Proceedings of the 6th European Conference on Planning (ECP01)*, 2001.
14. E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135, 1974.
15. D. E. Smith and M. A. Peot. A critical look at knoblock's hierarchy mechanism. In J. Hendler, editor, *Proceedings of the 1st International Conference on Artificial Intelligence Planning Systems(AIPS92)*, pages 307–308. Kaufmann, San Mateo, CA, 1992.
16. B. W. Wah and Y. Chen. Partitioning of temporal planning problems in mixed space using the theory of extended saddle points. In *15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2003)*, pages 266–273, 2003.
17. D. E. Wilkins. *Practical Planning: extending the classical AI paradigm*. Morgan Kaufmann, 1988.

Abstract Policy Evaluation for Reactive Agents

Krysia Broda and Christopher John Hogger

Department of Computing, Imperial College London,
South Kensington Campus, London SW7 2AZ UK
{kb, cjh}@doc.ic.ac.uk

Abstract. This paper describes a method for constructing and evaluating teleo-reactive policies for one or more agents, based upon discounted-reward evaluation of policy-restricted subgraphs of complete situation-graphs. The combinatorial burden that would potentially ensue from state-perception associations can be ameliorated by suitable use of abstractions and empirical simulation results indicate that the method affords a good degree of scalability and predictive power. The paper formally analyses the predictive quality of two different abstractions, one for applications involving several agents and one for applications with large numbers of perceptions. Sufficient conditions for reasonable predictive quality are given.

1 Introduction

Teleo-reactive (TR)-agents were introduced in [16] and further developed in [1] and [18]. Such agents act in response to stimuli received from their environment in such a way to predispose them towards achieving known goals. Their simplest program structure is a set (called a *policy*) of mutually-exclusive production rules of the form $perception \rightarrow action$, usually intended to control durative behaviour: given a current perception the agent performs the corresponding action until acquiring a new perception, whereupon it reacts likewise. We make two key assumptions about TR-agents: they have (i) little or no access to cognitive resources, such as beliefs or reasoning systems, and (ii) only partial observational capability, in that their perceptions may not capture the whole environmental state. A policy identified on this basis is *implicitly* goal-oriented. A significant advantage is the relatively low resources a TR-agent needs for its internal logic; unlike a deliberative agent [13] it does not need computational facilities capable of executing complicated software. Moreover, since the agent's policy is designed to be effective whatever the state in which it finds itself, unexpected exogenous changes in the environment do not cause difficulties. A framework for evaluating policies was proposed in [3] and extended in [4, 5] to use abstraction to deal with scalability, especially in multi-agent contexts. This paper investigates in Theorems 1 and 2 the level of approximation entailed in using abstractions by giving some sufficient conditions for reasonable predictive quality.

Our work is similar to, but different in approach from, those who seek to optimize simple agents, comparable to our own, by the use of Markov Decision

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 44–59, 2005.

Processes (MDPs) or – when the agents cannot perceive the state's entirety – Partially Observable MDPs (POMDPs) [7, 10, 15]. The key assumption made in these design methods is that beliefs about the agent's current state can be inferred on the basis of its previous action and/or current perception together with beliefs about its previous state, thence enabling a suitable next action to be chosen. This assumption yields algorithms capable of identifying policies that are optimal or near-optimal relative to one's ability to estimate probabilities given the agent's assumed powers of state observation. These methods are very successful when the above key assumption holds, but are more complicated to apply in the multi-agent context where the updating of each agent's beliefs has to consider the combinatorial impact of the other agents' actions upon the state. Our species of TR-agents are also different from those envisaged by [16], where the design of a good policy rests on the assumption that the goal state is totally observable. The content and ordering of the rules constituting the desired policy are inferred by a reductive planning process that constructs and orders rules so that the operation of each one may suitably enable the operation of others, the whole intended to ensure that the goal state eventually becomes achievable. We would also contrast our approach with those methods [9, 12, 14, 19, 21] that rely upon learning. Here the evolving experience of the agent is effectively translated into merit-oriented weightings of the alternative actions available to each perception. The outcome is typically a non-deterministic policy allowing the agent to choose, for its current perception, between alternative actions according to the weightings, which may be interpreted as the relative probabilities of those actions being the best to perform.

The next section describes our framework and presents two abstractions. Subsequent sections detail each kind of abstraction and analyse the approximations imposed on policy evaluation. The paper concludes with a discussion of the ramifications of our results.

2 Overview of Framework

Any world in which our agents operate is capable of assuming various *states*. An agent has three main features: a set $\mathcal{P}$ of *perceptions* it may have of its environment, a set $\mathcal{A}$ of *actions* it may take and a *policy* relating actions to perceptions. We here restrict the language of states, perceptions and actions to be propositional. In any state $o \in \mathcal{O}$, the agent's possible perceptions form some subset $P(o) \subseteq \mathcal{P}$. A *situation* is any pair (o, p) for which $o \in \mathcal{O}$ and $p \in P(o)$. We call the tuple $\langle \mathcal{O}, \mathcal{P}, \mathcal{A} \rangle$ a TR-application. A perception does not, in general, capture the entire world state and the agent normally perceives only limited information about that state. The problem is therefore how to find an optimal policy for a given goal for an agent that (generally) cannot recognize it.

2.1 Situation Graphs

Our framework is based upon a structure called the *unrestricted situation graph* G, which shows the situations that a representative agent called *self* may be

in and the possible actions it may take. Each directed arc in G is labelled by some action. When the agent is in a situation (o, p) its possible actions depend only upon p and form a set denoted by $A(p)$. A key feature of our framework is the process of pruning selected arcs from G according to some policy f, to leave the *f-restricted* graph, denoted by G_f. This graph commits the agent to take, in any situation, the action determined by policy f, and shows what will actually happen. We assume that every node in G_f other than a goal situation has a successor, possibly itself. Goal situations are not given a successor, as we are primarily interested in the effectiveness of policies to reach a particular goal and not necessarily in what happens afterwards. These things are summarised in Definition 1 and illustrated in Example 1 using *BlocksWorld*. (Of course, *BlocksWorld* is just an exemplar of a wide range of state transition systems.)

Definition 1. *Let $\langle \mathcal{O}, \mathcal{P}, \mathcal{A} \rangle$ be a TR-application. The* unrestricted situation graph*, denoted by G, is a directed graph whose nodes are all the acceptable situations admitted by the given application. A policy f is a total function from $\mathcal{P}$ to $\mathcal{A}$ and the* restricted situation graph*, denoted by G_f, is the result of pruning all arcs from G except those sanctioned by policy f.*

Example 1. There are 2 blocks on a table and an agent may see either the table ($s0$), or a block ($s1$), or a 2-tower ($s2$) if it exists, and may be holding (h), or not holding (nh), a block. The state is a list of the heights of towers present on the table. (Situations $4a$, $3d$ and $3e$ are possible only if there are several agents and are used in Example 2.) An agent may take one of the actions: `wander` (`w`), `pick` (`k`) or `put` (`t`). See Figure 1. The goal for this example is that at least one agent shall reach state 2 and see the 2-tower (i.e. be in situation $2c$). There is no action prescribed for perception c, since it occurs only in the goal situation. In what follows we will consider the policies Policy 1 and Policy 2, where Policy 1 always takes the `wander` action and Policy 2 is given by $a \rightarrow \mathtt{w}, \quad b \rightarrow \mathtt{t}, \quad d \rightarrow \mathtt{w}, \quad e \rightarrow \mathtt{w}$. Figure 2 shows restricted graphs for these two policies, in which all actions are `wander` except as indicated. The `wander` action is special in that it permits an agent to change its perception without a state change. Depending on the level of abstraction of the model the result of `wander` taken from situation s may, or may not, include s. In this example we assume it cannot be s.

	States
1	[1, 1]
2	[2]

	States
3	[1]
4	[ ]

	p	O(p)	A(p)
a	s0, h	{3, 4}	{t,w}
b	s1, h	{3}	{t,w}
c	s2, nh	{2}	{ }

	p	O(p)	A(p)
d	s1, nh	{1, 3}	{k,w}
e	s0, nh	{1, 2, 3}	{w}

Fig. 1. States, Situations and Actions for Example 1

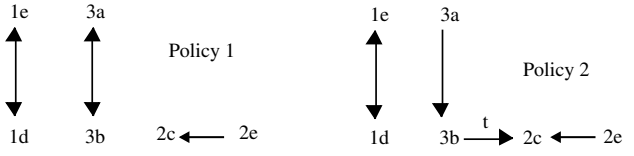


Fig. 2. Policies 1 and 2 (Example 1)

2.2 Measuring Policy Values

We measure the value of a policy f by the method of discounted rewards [10].

Definition 2. *Let f be a policy for a TR-application $\langle \mathcal{O}, \mathcal{P}, \mathcal{A} \rangle$, let $s = (o, p)$ be a situation in G_f and SS be the successor set of s. The* discounted reward *$V(s, f)$, effectively measuring the benefit of the agent proceeding from s, is given by the formula $V(s, f) = \Sigma_{u \in SS}(\chi_{su} \times (\Upsilon_{su} + \gamma \times V(u, f)))$.*

In the above, Υ_{su} is the immediate reward for the action that takes s to u, χ_{su} is the probability that from s the agent proceeds next to u and the factor γ discounts the benefit of taking that action at s. We choose $0 < \gamma < 1$ to reflect the diminishing returns to the agent of performing successive actions. Since we are interested in policies that perform well, on average, from whatever state an agent may find itself, these values of V are used to compute the overall value of f, denoted by $V_{\text{pre}}(f)$, given by the average of $V(s, f)$ taken over all nodes s in G_f. We distinguish two reward values: R for an arc leading immediately to a goal situation and r for all other arcs in G_f. The situations' values are related by a set of linear equations which, since $\gamma < 1$, have unique finite solutions.

There are two issues of scalability for which we propose abstractions. The first occurs when there are several agents. If every combination of agents were to be represented, then each situation would need to include each agent's perception. For applications with up to m perceptions and n agents this could potentially expand the number of situations and policies by a factor of n^m. We choose to approximate the restricted graph by focusing on the actions of a single agent called *self* (see[5]). Ramifications of the behaviour of other agents, necessarily in the same state as *self* but possibly having different perceptions, are reflected in G_f by the use of *exogenous arcs* (denoted by `x`-arcs). The second issue of scalability arises when the environment's size is increased – for example if there are many blocks. The increase in the number of states is usually accompanied by a gain in the number of perceptions and if every possible perception were to be represented even a small increase in G leads to a large increase in the number of policies. For example, having 10 blocks and allowing a single agent to have the 11 distinct perceptors $s0, \ldots, s10$ would give the agent 21 perceptions in all and one million policies to consider. Results presented in [6] show that both approximations still give reasonable predictions of relative policy values.

3 Formulation for Several Agents

If one were to use our framework to explicitly represent all situations for a group of n agents, the situations would necessarily consist of a state and an n-tuple of perceptions. This is the approach taken in [15], for example. Even for the simple case of *BlocksWorld* with 2 agents and 2 blocks, this gives 17 situations as opposed to 6 situations for a single agent. Nevertheless, we could imagine (but not actually construct) such an unrestricted situation graph; we call it the *group graph* denoted $\mathcal{G}^g$. In [5] for several agents of the same kind (i.e. having the same policy and called *clones*), we introduced the *self graph*, denoted $\mathcal{G}^s_f$, which focuses on a single agent. This graph is a projection of the group graph over the first (or any other) agent for a given policy and we showed it could be used to predict a good *joint policy*. Here, we do not require agents to be clones and instead call the various self graphs *viewpoint graphs*, denoted $\mathcal{G}^v$. It is desirable that the joint policy value should be approximated well by the policy value obtained for any single agent viewpoint. We illustrate for two small examples and in the following section consider under what restrictions the values of policies might be invariant when taken from the viewpoint any individual agent. The notion of TR-application is extended to allow for more than one kind of agent. We use the notation $\langle \mathcal{O}, \mathcal{P}, \mathcal{A}, \mathcal{R} \rangle$, where $\mathcal{R}$ is a set of one or more policies and each agent follows one of them (not necessarily uniquely). We assume here that all agents possess similar perceptive capabilities, although that need not always be so.

Definition 3. *Let $\langle \mathcal{O}, \mathcal{P}, \mathcal{A}, \mathcal{R} \rangle$ be a TR-application with n agents. The list $[(o, p_1), \ldots, (o, p_n)]$ is a* valid group *if in state o it is simultaneously possible for each agent i to have perception p_i. The set $\mathcal{S}_g$ of possible situations is given by $\mathcal{S}_g = \{(o, p_1, \ldots, p_n) | [p_1, \ldots, p_n]$ is a valid group for the n agents $\}$. The set $\mathcal{S}_g$ forms the nodes of the group graph $\mathcal{G}^g$ and its transitions $\mathcal{T}_g$ are derived from the possible transitions any agent could make from each situation: $s = (o, p_1, \ldots, p_n)$ is connected to $s' = (o', p'_1, \ldots, p'_n)$ by action a^i if some agent i in the group can take the action a^i in s to bring about s'. In particular, agent i, when in the individual situation (o, p_i) and taking action a^i, causes itself to make a transition to (o', p'_i) and other agents to their perceptions given by s'.*

That is, each valid group of simultaneous perceptions gives rise to a situation in the group graph and each valid transition of a single agent gives rise to a transition in the group graph. The probabilities on each transition are proportional to those of the individual transitions; *e.g.* if Agent 1 has a non-deterministic action from some situation (o, p_1) with two equi-probable outcomes, then if there are 3 agents these transitions would each have probability 1/6 from any group situation (o, p_1, p_2, p_3). When there are several agents it is possible that *self*'s best policy is to wait for some other agent to change the state, whence it continues. We introduce the `x` action for this purpose. To obtain a viewpoint graph from a group graph, first a particular policy is fixed for each agent and a restricted group graph formed by omitting all arcs except those of the policy given for each agent. Then a projection of the restricted group graph is taken from the point of view of a particular agent i. It is also possible that, from the view of *self*, the

state is exogenously changed by another agent. We call this *passive updating* of *self* and label such transitions (in the viewpoint graph) also by `x`.

Definition 4. *Let $\langle \mathcal{O}, \mathcal{P}, \mathcal{A}, \mathcal{R} \rangle$ be a TR-application with n agents and $\mathcal{G}^g$ be a restricted group graph based on the set $\mathcal{S}_g$ of situations of the form $(o, p_1, \ldots, p_n)$ and having set of transitions $\mathcal{T}_g$. Then $\mathcal{G}_i^v$ is the* viewpoint graph *for agent i obtained from $\mathcal{G}^g$ as follows. The situations of $\mathcal{G}_i^v$ are projections of those in $\mathcal{G}^g$ and have either the form (a) (o, p_i), in case $(o, p_1, \ldots, p_n)$ is not a goal situation of $\mathcal{G}^g$, or the form (b) (o, p_j) in case it is, where agent j is responsible for $(o, p_1, \ldots, p_n)$ being a goal situation. The set $\mathcal{T}_v$ of transitions in $\mathcal{G}_i^v$ is given by $\mathcal{T}_v = \{((o, p_i), (o', p_i'))\}$, where $(o, p_1, \ldots, p_i, \ldots, p_n)$ to $(o', p_1', \ldots, p_i', \ldots, p_n')$ is a transition in $\mathcal{G}^g$ and the action for a transition not due to the action of agent i is* `x`*, and otherwise is the action taken by agent i.*

A situation in a viewpoint graph may correspond to several situations in the group graph from which it is derived. The *abstraction function ab*, a mapping from situations in $\mathcal{G}^g$ to situations in $\mathcal{G}^v$, records the correspondences and induces an equivalence relation E_a on the situations in $\mathcal{G}^g$. The E_a equivalence class of a situation s in $\mathcal{G}^g$, denoted $[s]$, is $\{s' | ab(s) = ab(s')\} = ab^{-1}(ab(s))$. In other words, the inverse images of situations in $\mathcal{G}^v$ are the E_a equivalence classes of the situations in $\mathcal{G}^g$. The transition probabilities for $\mathcal{G}^v$ (for Agent i) are derived in proportion to those in $\mathcal{G}^g$ as follows: for a transition between $s1$ and $s2$ in $\mathcal{G}^v$ due to an action a^i of Agent i, the sum of probabilities between each situation in $ab^{-1}(s1)$ and a situation in $ab^{-1}(s2)$ in $\mathcal{G}^g$ due to action a^i of Agent i is computed and divided by $|ab^{-1}(s1)|$. If there are `x`-arcs between $s1$ and $s2$ due to actions of some other Agent j, $j \neq i$ the sum of probabilities over all corresponding arcs between $ab^{-1}(s1)$ and $ab^{-1}(s2)$ is divided by $|ab^{-1}(s1)|$ to give the probability of an `x`-transition between $s1$ and $s2$. The sum of all resulting probabilities of arcs from $s1$ will be 1, since in $\mathcal{G}^g$ the probabilities summed to 1 for each situation in $ab^{-1}(s1)$.

Example 2 *(extends Example 1).* The group graph $\mathcal{G}^g$ is shown in Figure 3, including the various situations, in which the leftmost arrow indicates the status (either seeing the table or a block, and holding (H) or not) of Agent 1 using Policy 1 and the rightmost the status of Agent 2 using Policy 2. All probabilities are 0.5 unless shown otherwise and all actions are `w` except as indicated. There are 17 nodes; nodes 5, 6 and 17 are designated goal situations, when at least one agent is seeing the 2-tower (situation $2c$). The joint policy, obtained using $\mathcal{G}^g$, has the following approximate node values: $v(1) = v(2) = v(3) = v(4) = v(8) = v(9) = v(10) = v(16) = v(11) = -10$; $v(5) = v(6) = v(17) = 0$; $v(7) = v(12) = v(14) = 90$; $v(13) = v(15) = 59$ and total value of 298/14.

To form the viewpoint graphs we use abstraction maps ab_1 and ab_2 between situations in $\mathcal{G}^g$ and $\mathcal{G}^v1$ and $\mathcal{G}^v2$, which are shown together with the viewpoint graphs for the two agents in Figure 4. All probabilities are 0.5 unless indicated otherwise. From the view of Agent 2 there would initially appear to be no possible exogenous transitions to passively update Agent 2, for since Agent 1 can only `wander` it cannot alter the state. However, $\mathcal{G}^v2$ has a reflexive `x`-arc from $3b$ to

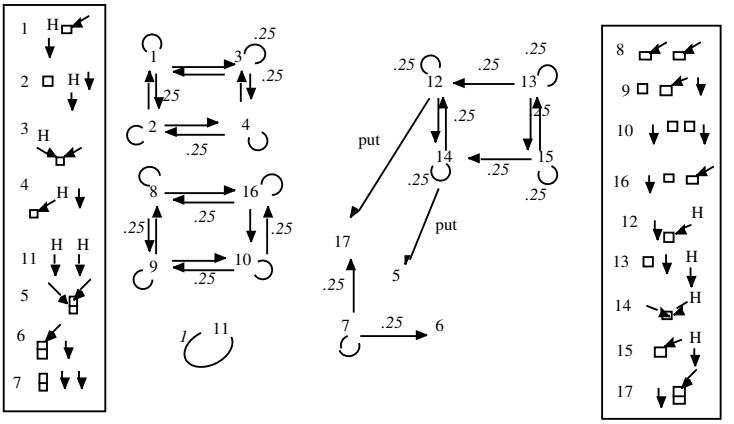


Fig. 3. Group Graph for two Agents using Policies 1 and 2 (Example 2)

s in Gg	s in Gv1	s in Gv 2
1	3a	3d
2	3a	3e
3	3b	3d
4	3b	3e
8	1d	1d
10	1e	1e
16	1e	1d
11	4a	4a
5	2c	2c
6	2c	2c
17	2c	2c
7	2e	2e
12	3e	3b
13	3e	3a
14	3d	3b
15	3d	3a

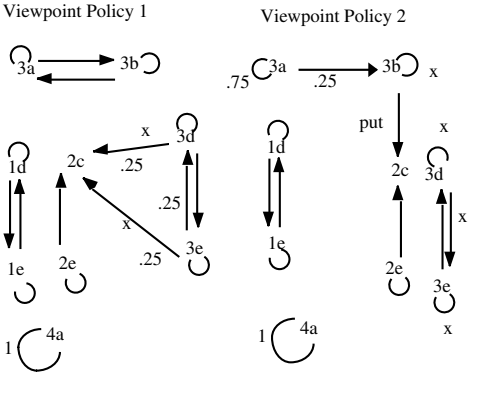


Fig. 4. Viewpoint Graphs $\mathcal{G}^v1$ and $\mathcal{G}^v2$ for Example 2

itself arising from the transitions in $\mathcal{G}^g$ between situations 12 and 14 due to the actions of Agent 1. We illustrate the computation for situation $3b$ in $\mathcal{G}^v2$. The arcs between situations 12 and 14 and the respective arcs on these nodes all arise from the `wander` action of Agent 1 and summing these probabilities in $\mathcal{G}^g$ gives 1. Similarly, the result of summing the probabilities on transitions in $\mathcal{G}^g$ between 12 or 14 and any of the goal situations, corresponding to a transition in $\mathcal{G}^v2$ between $3b$ and $2c$, is 1. The size of $ab^{-1}(3b)$ is 2, giving probabilities of 0.5 on both arcs from $3b$ in $\mathcal{G}^v2$. The correspondence between situation 6 and $2c$ for Agent 2 is obtained by case (b) of Definition 4. On the other hand, from the view of Agent 1, there are some obvious exogenous behaviours. When Agent 1 is in situation $3d$ or $3e$, then Agent 2 would necessarily be in $3a$ or $3b$ and, if in $3b$, Agent 2's action would be `put`, so constructing a 2-tower.

If the joint policy is now evaluated from $\mathcal{G}^v2$, the node values obtained are: $v(3b) = v(2e) = 90$, $v(3a) = 59$, $v(2c) = 0$ and other node values $= -10$. The total value is 189/8, quite close to the value obtained for $\mathcal{G}^g$. However, if a weighted average of the node values is taken, according to the number of elements in $ab^{-1}(s)$, for each s, the average is $(9 \times -10 + 2 \times 90 + 1 \times 90 + 2 \times 59 = 298/14$. If the joint policy is now evaluated from $\mathcal{G}^v1$, the node values obtained are: $v(2e) = 90$, $v(3d) = v(3e) = 74.6$, $v(2c) = 0$ and -10 for the remainder. The weighted average is also 298/14, again exactly the value of the joint policy obtained from the group graph. This desirable circumstance does not always prevail, as the next Example shows.

Example 3. This example is from *Planks World*, in which two identical agents aim to dispose of a plank, for which each must be holding a (different) end. This time the joint policy values for the group graph and viewpoint graphs differ. The states and situations are given in Figure 5. Each agent is capable of the actions `wander`, `drop`, `lift`, `x` and `dispose`. The situation $(0, a)$ is the goal and the states 1-3 are given by describing whether the single plank is (f)lat, (t)ilted or (r)aised. The agents can perceive whether they are holding an end (h) or not (nh), seeing a held or unheld end (sh or su) and, if holding, whether the plank is raised (r) or not raised (nr). It is assumed that an agent can see a held end if it is holding. Policy 3 specifies the following actions for each perception: $a \rightarrow$ `w`, $c \rightarrow$ `li`, $e \rightarrow$ `w`, $f \rightarrow$ `x`, $g \rightarrow$ `di` and the viewpoint graph (projected over Agent 1) and a fragment of the group graph are given in Figure 6, in which all actions are wander unless shown otherwise and unlabelled transition probabilities are 0.5. If the joint policy value is computed from the group graph, the approximate node values obtained are: $v(1a, 1a) = 35$; $v(1c, 1a) = v(1a, 1c) = 42$; $v(1c, 1c) = 55$; $v(0a, 0a) = 0$; $v(3g, 3g) = 100$; $v(2f, 2c) = v(2c, 2f) = 80$; $v(2f, 2a) = v(2f, 2e)) = v(2a, 2f) = v(2e, 2f) = 44$ with total policy value of 610. If instead the joint policy value is computed from the viewpoint graph $\mathcal{G}^v$, the node values obtained are: $v(1a) = 36$, $v(1c) = 50$, $v(2a) = v(2e) = 44$, $v(2f) = 56$, $v(2c) = 80$, $v(3g) = 100$ and $v(0a) = 0$ with approximate total weighted value of 608. (*e.g.* the probabilities of the two arcs incident to situation $2f$ are derived from the single transition to $(3g, 3g)$ (the one to $3g$) and the 9 transitions between situations $(2f, 2c)$, $(2f, 2e)$ and $(2f, 2a)$ (the reflexive arc). In fact, the `wander` arcs contribute 1/3 and the `x`-arcs contribute 0.5 to the reflexive arc.) Although the two values obtained for the joint policy are very close, they are not equal. In the next section we give criteria which are sufficient to force the two computations to give identical values. These criteria are satisfied for Example 2, but not for Example 3.

	States
0	[ ]
1	[f]

	States
2	[t]
3	[r]

	p	O(p)	A(p)
a	s0, nh	{0, 1, 2}	{`w`, `x`}
c	su, nh	{1, 2}	{`li`, `w`, `x`}
e	sh, nh	{2}	{`w`, `x`}

	p	O(p)	A(p)
f	sh, h, nr	{2}	{`dr`, `x`}
g	sh, h, r	{ 3}	{ `di`, `dr`, `x`}

Fig. 5. States, Situations and Actions for Example 3

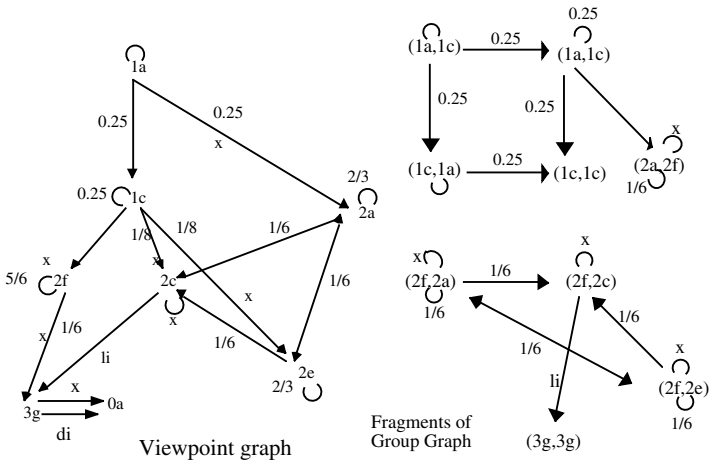


Fig. 6. Graphs for Example 3

4 Relationship Between Group and Viewpoint Graphs

Examples 2 and 3 above have shown that the policy values of a group graph and viewpoint graphs derived from it need not be equal. In Example 3, the value of node $2f$ (56) was exactly one-third of the sum of the values of the three nodes in the equivalence class $ab^{-1}(2f)$ (80+44+44). This isn't a coincidence, but does not always obtain; for instance, also in Example 3, the value of node $1c$ (50) is *not* one-half the sum of the values of the two nodes in $ab^{-1}(1c)$ (55+42). Theorem 1 states some sufficient conditions for the above property to hold.

Theorem 1. *Let $\mathcal{G}^g$ be a group graph and $\mathcal{G}^v$ be the viewpoint graph for one of the Agents. Let s be a situation in $\mathcal{G}^v$ and N be the set of situations in $\mathcal{G}^g$ that are mapped to s by ab. Assume also that the rewards on arcs directed to nodes in the same equivalence class of $\mathcal{G}^g$ are equal. Then the quantity $v(s) \times |N|$ is equal to $\Sigma_{n \in N} v(n)$ if either of the following two circumstances holds.*

(i) For each m in $\mathcal{G}^g$ not in N and not of type (ii), and for which there is an arc to m from some node in N, there is an arc from every node in N to every node in $[m]$, the E_a equivalence class of m, all with the same probability, and either the probabilities on all those kind of arcs are equal for every node in $[m]$ or every node in $[m]$ has equal value.

(ii) For each m in $\mathcal{G}^g$ not in N such that the E_a equivalence class of m is a singleton, exactly one node n in N has an arc leading to m, which is n's only non-reflexive arc, all other arcs from situations in N lead to other nodes in N and the probabilities on these are in certain proportions: for each $n' \in N$, $n' \neq n$ the probability of the reflexive arc must be $pr + p_{n'}$, where pr is the same for all n and the probability of transitions between other nodes in N and n' is $p_{n'}$.

Proof. If N is the set of situations in $\mathcal{G}^g$ that are mapped by ab to s in $\mathcal{G}^v$, we shall write $ab(N) = s$. We shall also write $v(s)$ for the value of a situation as computed by the discounted reward formula. The conditions on (i) imply that if any node in N satisfies them, then each node in N satisfies them and except in the trivial case, when N and $[m]$ are both singletons, the two cases are disjoint. Let the size of N be k and each node in N have a bundle of arcs to nodes m belonging to some set DM. We shall denote the value of $ab([m])$ in $\mathcal{G}^v$ by vM and the reward for reaching a node in N or $ab(N)$ by rN and for reaching a node in $[m]$ or $ab([m])$ by rm.

Case (i). Each node in DM is of the type described in (i), hence for each node $m \in DM$ the probability p_{nm} of the arc directed to m from $n \in N$ is the same. The sum of the probabilities of arcs from a node $n \in N$ to other nodes in N is $1 - \Sigma_{m \in DM}(p_{nm})$ and in the viewpoint graph $\mathcal{G}^v$ the probability from $ab(N)$ to itself is $1 - K$, where $K = \Sigma_{m \in DM}(p_{nm})$. There are two sub-cases:

(a) In $\mathcal{G}^v$ the probability of the transition from $ab(N)$ to $ab([m])$ is K_m, where K_m is $\#[m] \times p_{nm}$, where p_{nm} is the same probability for the transition in $\mathcal{G}^g$ from each node in N to each node in $[m]$ and $\#[m]$ is the size of $[m]$. The values of the nodes in $[m]$ may be different, but their mean is Vm.
(b) In $\mathcal{G}^v$ the probability of the transition from $ab(N)$ to $ab([m])$ is $\Sigma_{m' \in [m]} K_{m'} = K_m$, where $K_{m'}$ is the same probability for the transition in $\mathcal{G}^g$ from each node in N to m'. The values of the nodes in $[m]$ are equal and denoted by Vm.

Case (a). Using the discounted reward formula on $ab(N)$ $(= s)$ in $\mathcal{G}^v$ gives $v(s) = \gamma(1-K)v(s) + \gamma\Sigma_{m \in ab(DM)}(K_m.Vm) + (1-K)rN + K.rm$, and hence $v(s) = (\gamma\Sigma_{m \in ab(DM)}(K_m.Vm) + (1-K)rN + K.rm)/1 - \gamma(1-K)) = C/(1 - \gamma(1-K))$, where $C = \gamma\Sigma_{m \in ab(DM)}(K_m.Vm) + (1-K)rN + K.rm$.

The values in $\mathcal{G}^g$ of nodes in N can be computed as the sum of the contributions derived from transitions to nodes in N, denoted by $rest_n$, and other transitions (to nodes in DM). The probability from a node in N to each node in $[m]$ is $K_m/\#[m]$. The second contribution is the same for all nodes in N and is given by $\gamma\Sigma_{[m] \subseteq DM}((K_m/\#[m])\Sigma_{m' \in [m]} v(m')) + K.rm + (1-K)rN = C$.

It is required to show that $\Sigma_{n \in N} v(n) = k \times v(s)$. For each $n \in N$, $rest_n = \gamma\Sigma_{n' \in N}(p_{nn'}.v(n')) = \gamma\Sigma_{n' \in N}(p_{nn'}(C + rest_{n'})) = \gamma C\Sigma_{n' \in N}(p_{nn'}) + \gamma\Sigma_{n' \in N}(p_{nn'}.rest_{n'}) = \gamma C(1-K) + \gamma\Sigma_{n' \in N}(p_{nn'}.rest_{n'})$.

Now, $\Sigma_{n \in N} v(n) = kC + \Sigma_{n \in N} rest_n$, where $\Sigma_{n \in N} rest_n = k\gamma.C(1-K) + \gamma(1-K)\Sigma_{n \in N} rest_n$. This follows since the full expression for $\Sigma_{n \in N} rest_n$ has a unique solution (and one solution is to set all $rest_n$ equal). Hence $\Sigma_{n \in N} v(n) = kC + (k\gamma C(1-K)/(1 - \gamma(1-K)) = kC/(1 - \gamma(1-K))$.

Case (b). This time the value of $v(s)$ is $\gamma(1-K)v(s) + (1-K)rN + K.rm + \gamma\Sigma_{m \in ab(DM)}(K_m.Vm)$ and hence $v(s) = C/(1 - \gamma(1-K))$.

In the group graph $\mathcal{G}^g$ the probabilities of transitions from nodes $n \in N$ to each node $m' \in [m]$ are the same, denoted by $K_{m'}$, although they may be different for each m'. Then, for a node $n \in N$, $v(n) = \gamma\Sigma_{[m] \subseteq DM}(\Sigma_{m' \in [m]}(K_{m'}.Vm)) +$

$K.rm + (1-K)rN + rest_n = \gamma\Sigma_{[m]\subseteq DM}(K_m.Vm) + K.rm + (1-K)rN + rest_n$, where again $rest_n$ is the sum of contributions to $v(n)$ derived from arcs to nodes in N. The first three parts are the same for every n and their sum is C. It is again required to show that $\Sigma_{n\in N}v(n) = k \times vN$ and similar computations for $rest_n$ can be made as before, giving the result.

The proof of **Case (ii)** is simpler and makes similar sorts of calculations. □

Example 2 meets the criteria of Theorem 1 whereas Example 3 does not. For instance, consider $\mathcal{G}^v1$ of Example 2. For $s = 3a$ and $N = \{1, 2\}$, $[m] = \{3, 4\}$ and case (ia) is satisfied. For $s = 3d$ and $N = \{14, 15\}$, $[m]$ can be either $\{12, 13\}$, and case (ia) is satisfied, or $\{5\}$ and case (ib) is satisfied. On the other hand, in Example 3 criteria (ii) is satisfied for $2f$: $pr = 0.5$ and $p_n = 1/6$ for both $(2f, 2a)$ and $(2f, 2e)$. For $1a$ it is not satisfied, as the reader can easily check.

Although seemingly restrictive, in practice the restrictions on probabilities are often nearly satisfied. Even when not, the proof method shows that, unless the relevant probabilities are wildly variant, the two quantities will still be fairly close since the viewpoint policy averages the various probabilities as if they were equal. This result gives some foundation to our empirical results, obtained in [6], which show that the ranks of the viewpoint policy values, computed using $\mathcal{G}^v$, *are* a good guide to the ranks of the group policy values, computed using $\mathcal{G}^g$.

If the criteria are not met some variation should be expected between the joint policy value as computed by the group and viewpoint graphs. In particular, paths may exist in the viewpoint graph that are not realizable. The viewpoint graph of Example 3 contains the path $\{1a, 1c, 2e, 2c, 3g, 0a\}$, which abstracts the real path $\{(1a, 1c), (1c, 1c), (2e, 2f), (2c, 2f), (3g, 3g), (0a, 0a)\}$, but also implicitly includes impossible paths. For instance, after starting from the possible transition $\{(1a, 1a), (1c, 1a)\}$ Agent 1 cannot move to $2e$. In this example every path in the viewpoint graph corresponds to at least one path in the group graph, but if there are 2 planks and 2 agents, then paths can be found in the viewpoint graph that do *not* correspond to any realizable path. The viewpoint graph has abstracted away details of the groups, and although there may be arcs leading through situations $\{s1, s2, s3\}$ the group that occurs as a result of the first transition may not be a correct one from which to make the second. We call this the *group incoherence problem*. In extreme cases, the valuation of nodes that apparently, but incorrectly, lead to a goal situation can inflate the policy value, so that a bad policy appears better than it really is. On the other hand, the extreme case appears to be fairly rare, so the benefit of viewpoint graphs for scalability outweighs the disadvantages due to the group incoherence problem.

5 Policy Abstraction

There is another way to obtain abstractions. Consider an agent operating in a *BlocksWorld* with many (e.g. 10) blocks; assuming just the actions `wander`, `pick` and `put` there are 21 perceptions and over a million policies to evaluate. The

number of policies can be reduced by generalising the perceptions. For example, it could be that an agent can, and only needs to, detect a tower of height $= 0, < 5$ or $= 5$ or > 5. This generalisation can be seen either to be an enhancement of the capabilities of the agent, for example by allowing it to perceive disjunctions, as in seeing a tower of height 1, 2, 3 or 4 (the perception < 5), or by giving it more power to sense the height of a tower; or to be an increase in expressiveness of the policy language, for instance by using first order logic and allowing perceptions of the form $\{size(x), x < 5, x > 0\}$. Either way, not only do the perceptions need to be abstracted, but also several states may need to be combined in order for situations of the form (o, p) to be meaningful. In [4] we investigated this kind of abstraction, and our simulation studies showed it still gave good ranking charts for policies in cases where the number of policies was too large for individual computation. The discrepancies are again due to a coherence problem and to explain it we consider what approximations are involved in calculating policy values for such abstractions.

5.1 Generic Situations

From any TR-application $\langle \mathcal{O}, \mathcal{P}, \mathcal{A} \rangle$, we can form a *generic TR-application*, in which the actions remain unchanged, but the states and perceptions are generalised, which means to introduce, respectively on $\mathcal{P}$ and $\mathcal{O}$, equivalence relations E_p and E_o. The $E_p(E_o)$ equivalence classes are called *generic perceptions(states)* and if perceptions p_1 and p_2 are E_p *equivalent* they will always specify the same action. We could require that an agent is capable of taking the same actions for all perceptions in each E_p equivalence class, or that the policy specifies such an action, but it is not necessary, since if, for some generic perception P, the action specified is not possible for an actual perception in P it could be modelled by a failed action. For each generic state O and generic perception P the *generic situation* $S = O \times P$ is disjoint from all others, which is important since it ensures that no policy can specify two different actions for any real situations.

Example 4. This can be illustrated straightaway for *BlocksWorld* by using a generalisation with just two generic states $[e3]$ and $[ne3]$, denoted by 1 and 2, and three seeing perceptors, $s0$, $s3$ and sx, the latter denoting "seeing neither the surface nor a 3-tower". This yields the 6 perceptions $a - f$ given by $\{(s0, h), (sx, h), (s3, h), (s0, nh), (sx, nh), (s3, nh)\}$. This abstraction suits the goal of building a 3-tower from an arbitrary but sufficient number of blocks. The situations and transitions for the policy whereby the agent, if seeing a tower of height neither 0 nor 3 (sx), can `pick` if not-holding (nh) or `put` if holding (h), but in all other cases wanders, is shown for 3 blocks in Figure 7 (perception c is impossible). The intended goal is the situation $1f$ (i.e. $([e3], (s3, nh))$).

A comparison with any standard restricted graph G for 3 or more blocks shows a second incoherence problem called *piecewise incoherence*. In the generic restricted graph there is a path through situations $\{2e,\ 2b,\ 1f\}$. However, this path could never actually occur – both parts of the path from $2e$ to $1f$ are possible, but not in succession. The situation $2e$ corresponds to an agent seeing

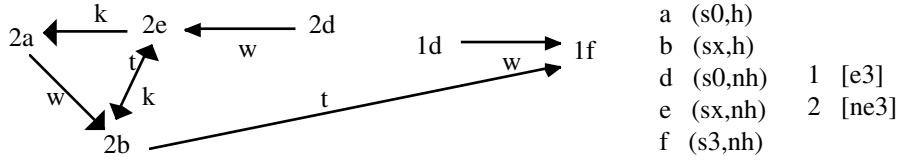


Fig. 7. Using generic situations (Example 4)

a tower of height 1, 2 or $\geq$4 and no 3-tower in existence. The policy specifies the `pick` action causing the agent to move to 2*b*. In fact, that means the agent could not have been seeing a tower of height 1 or of 4 when in situation 2*e* and the agent must now be seeing a tower of height 1 or $\geq$4, for which the policy specifies a `put` action. The outcome of this action in this particular circumstance could never be 1*f*. In G there would be nodes from which a 3-tower cannot be built due to `pick` and `put` actions between situations where the agent sees a tower of height 2 or $\geq$5 and is not holding. For 3 blocks and the above policy G has a value of 51.6, whereas the graph in Figure 7 has a value of 71.5 (where node values are weighted by the number of concrete situations represented by each generic situation and assuming equi-probable transitions in both cases).

The reader may think the problem could be overcome by enhancing the agent with an extra sense, *e.g.* allow it to recall its previous action, so distinguishing between having arrived at 2*b* via 2*e* or via 2*a*. This splits the *b* perception into two, one in which the agent remembers its previous action was `wander`, and one in which it remembers it was `pick`, but it results in non-disjoint generic situations unless a similar perceptive capability obtains in the standard graph G and illustrates the care that must be taken when constructing generic graphs.

5.2 Evaluating Generic Policies

This section discusses the relation between the policy value of a generic graph, and the policy value of the non-abstracted graph for the corresponding policy. The analysis made in the proof of Theorem 2 will also yield a criterion that guarantees no piecewise incoherence in a generic graph.

Definition 5. *Let* $\langle \mathcal{O}, \mathcal{P}, \mathcal{A} \rangle$ *be a TR-application and* E_p *and* E_o *be equivalence relations on the sets* $\mathcal{O}$ *and* $\mathcal{P}$ *respectively. Then* $\langle E_o, E_p, \mathcal{O}, \mathcal{P}, \mathcal{A} \rangle$ *is the* generic TR-application based on $\langle \mathcal{O}, \mathcal{P}, \mathcal{A} \rangle$ *and the set of generic situations is* $\{S | S = O \times P\}$*, where* O *and* P *are, respectively,* E_o *and* E_p *equivalence classes.* $\langle \mathcal{O}, \mathcal{P}, \mathcal{A} \rangle$ *is called the parent application.*

The elements of a generic situation that also exist in its parent application are called *concrete situations*. The probability of a transition from $S1$ to $S2$ in a restricted generic graph G_a is computed as the mean of the probabilities of transitions from a concrete situation in $S1$ to a concrete situation in $S2$

and is denoted by χ^a. We also make two assumptions: (a) the rewards on any transition leading to an element of a generic situation $S1$ are the same, and (b) if generic policy F specifies $P \rightarrow a$, then the corresponding policy f for the parent application specifies the rules $p \rightarrow a$ for every $p \in P$. Assumption (a) imposes the restriction on the equivalence classes E_p and E_o that goal situations and non-goal situations cannot be equivalent.

Theorem 2. *Let G_a be a restricted graph for a generic TR-application and G_f be the restricted graph for the parent application using corresponding policy f. Then, for each generic situation S in G_a, $k \times V_S = \Sigma_{i \in S} v_i$ if the probabilities on each transition between S and U in G_a are the average of the transition probabilities between each $i \in S$ and $j \in U$. (V_S and v_i are the values of situations S and i in G_a and G_f and k is the number of concrete situations in S.)*

Proof. For simplicity, zero probabilities are assigned to non-existent transitions. Let S be a generic situation in G_a. Then the sum of values of concrete situations in S is given by $\Sigma_{i \in S} v_i = \gamma \Sigma_{i \in S} (\Sigma_{j \in U, U \in G_a} (\chi_{ij} . v_j)) + \Sigma_{i \in S} (\Sigma_{j \in U, U \in G_a} \chi_{ij} . r_{ij})$, where r_{ij} is the reward on the transition between i and j if it exists (and is irrelevant otherwise). By the assumption (a) each of r_{ij} is the same for and $j \in U$ and all $i \in S$, so the contribution due to reward values can be simplified to $\Sigma_{U \in G_a} (r_{SU} \Sigma_{i \in S, j \in U} \chi_{ij})$.

The quantity $k \times V_S$ is given by $k\gamma \Sigma_{U \in G_a} (\chi^a_{SU} . V_U) + k \Sigma_{U \in G_a} (\chi^a_{SU} . r_{SU})$, The contributions due to reward values are the same in both cases for each U since $k \times \chi^a_{SU} = k(\Sigma_{i \in S, j \in U} \chi_{ij})/k$. The other contribution to $\Sigma_{i \in S} v_i$ can be written as $k\gamma \Sigma_{j \in U \in G_a} (\chi_{Sj} v_j)$, where χ_{Sj} is the mean of the transition probabilities between each concrete situation in S and each concrete situation $j \in U$. If the χ_{Sj} are further averaged over each j, each to be equal to χ_{SU}, then the contribution becomes $k\gamma(\Sigma_{U \in G_a} \chi_{SU} \Sigma_{j \in U} v_j)$. By comparing the expressions for $k \times V_S$ and $\Sigma_{i \in S} v_i$, it can be seen they would be equal if V_S were the average of v_i, $i \in S$. □

In other words, the policy value using generic situations is obtained by assuming that the transition probabilities between generic situations are an average of the actual transition probabilities and that the node values are an average of the concrete situation values. Therefore, in the case that some transitions do not exist, piecewise incoherence is a possibility. In cases where the transition probabilities between the concrete situations making up two generic situations vary widely and/or the values of the concrete situations also vary widely, the generic policy will not be a good reflection of the policy using concrete situations.

In Example 3, notice that if there are many blocks then the probability of situation $2b$ occurring when the agent is seeing a tower of height > 4 is much increased. Thus the probability of the arc between $2b$ and $1c$ would, in practice (i.e. if measured by simulation), be quite small compared with that of the arc between $2b$ and $2e$. This would cause a corresponding reduction in the policy value and improve the approximation, since the generic policy value over-estimates the contribution to the value of $2b$ made by the arc to $1c$. The

absolute policy values are not as crucial as their relative ranking – even if the policy values computed using G_a are higher than those computed using the full graph G_f, if the values are ranked in the same order in both cases this will still allow for the best policies to be found. In experiments conducted so far (see [4, 6]) this has been the case. This abstraction has some similarities with that introduced in [11], where a theorem similar to Theorem 2 is quoted. In the circumstances when a transition in G_a between (O, P) and (O', P') implies there is a transition between every $(o, p) \in (O, P)$ and $(o', p') \in (O', P')$ generic policies always give reasonable approximations. Piecewise coherence cannot then occur, since the destination in (O', P') of a concrete transition from (O, P) would always be a source for the next transition from (O', P') to some other generic situation. Abstractions satisfying this criterion were considered in [17] and arise naturally when the goal situation is also changed to reflect the changes in the environment due to scaling; *e.g.* in *BlocksWorld* this kind of goal might be to build a tower of all available blocks, or in *PlanksWorld* it might be to dispose of all planks however many there were initially.

6 Conclusions and Future Work

We have analysed the approximations involved in using abstractions to evaluate policies for TR-agents, in order to test the predictive quality of such abstractions in contexts involving several agents and/or many situations. In the case of several agents we approximated the group behaviour by focusing on a single agent and Theorem 1 shows that the policy values are generally affected and may be subject to group incoherence. This phenomenon is more likely in case there are few states and many perceptions for each state; however, as we assume fairly simple agents, this circumstance appears to be relatively uncommon, which is borne out by our empirical studies in [5, 6]. Moreover, it is less likely in case of a large number of agents, since all perceptions of a given state will be more common, in turn making exogenous transitions in $\mathcal{G}^v$ more likely to occur in practice.

We are investigating the benefits obtainable when agents possessing different perceptive capabilities operate in the same environment. For example, some agents may be endowed with global perceptions, and be capable of few actions, whereas other agents may be capable of more specific actions and perceptions. The former kind of agent could act as an information source for other agents.

We also analysed the approximations due to perceptual abstractions and we found they could give rise to piecewise incoherence and that this was a more common phenomenon than group incoherence. However, empirical results in [4, 6] show that in case the transition probabilities are well estimated, the abstract policy values give fair relative predictions for exact policy values. The results depend on the particular generic situations chosen and this is a topic for our future investigation, together with a comparison of our work with that of [11]. well under changing environmental conditions.

References

1. Benson, S. Inductive Learning of Reactive Action Models, PhD, Dept. of Computer Science, Stanford University, 1996.
2. Benson, S. and Nilsson, N. Reacting, planning and learning in an autonomous agent, Machine Intelligence 14, Eds. Furukawa, K., Michie, D. and Muggleton, S, Clarendon Press, Oxford, 1995.
3. Broda, K. , Hogger, C.J. and Watson, S. *Constructing Teleo-reactive Robot Programs*, Proc. 14th European Conf. on A.I. (ECAI-2000), Berlin, pp. 653-657, 2000.
4. Broda, K. and Hogger, C.J. *Designing and Simulating Individual Teleo-Reactive Agents*, Poster Proceedings, 27th German Conf. on AI, Ulm, pp. 1-15, 2004.
5. Broda, K. and Hogger, C.J. *Policies for Cloned Teleo-Reactive Agents*, 2nd Conf. on Multi-Agent System Technologies, Erfurt, LNAI 3187, pp. 328-340, 2004.
6. Broda, K. and Hogger, C.J. *Determining and Verifying Good Policies for Several Cloned teleo-Reactive Agents*, Intelligent Systems Journal, MATES Special Issue, 2005 (to appear).
7. Cassandra, A.R., Kaelbling, L.P. and Littman, M. Acting Optimally in Partially Observable Stochastic Domains, *Proceedings 12th National Conference on AI (AAAI-94)*, Seattle, pp 183-188, 1994.
8. Chades, I., Scherrer, B. and Charpillet, F. Planning Cooperative Homogeneous Multiagent Systems using Markov Decision Processes, *Proc. of the 5th International Conf. on Enterprise Information Systems (ICEIS 2003)*, pp 426-429, 2003.
9. Dickens, L. Learning through Exploration, MSc Dissertation, Dept of Computing, Imperial College, 2004.
10. Kaelbling, L.P., Littman, M.L. and Cassandra, A.R. Planning and Acting in Partially Observable Stochastic Domains, *Artificial Intelligence* 101, pp 99-134, 1998.
11. Kersting, K. and De Raedt, L. Logical Markov Decision Programs and the Convergence of Logical TD(λ), *Proc. of ILP2004*, LNAI 3194, pp180-197, 2004.
12. Kochenderfer, M. Evolving Hierarchical and Recursive Teleo-reactive Programs through Genetic Programming, *EuroGP 2003*, LNCS 2610, pp 83-92, 2003.
13. Kowalski, R.A. and Sadri, F. *From logic programming to multi-agent systems*, in: Annals of Mathematics and Artificial Intelligence, 25, pp. 391-419, 1999.
14. Mitchell, T., *Reinforcement Learning*, in Machine Learning, McGraw Hill, 1997.
15. Nair R., Tambe, M., Yokoo, M., Pynadath, D. and Marsella, M. Taming Decentralised POMDPs: Towards Efficient Policy Computation for Multiagent Settings, *Proc. of the 18th Int. Joint Conf. on A.I. (IJCAI-03)*, pp 705-711, 2003.
16. Nilsson, N.J. *Teleo-Reactive Programs for Agent Control*, Journal of Artificial Intelligence Research, pp. 139-158, 1994.
17. Nilsson, N.J. *Learning Strategies for Mid-level Robot Control: some preliminary Considerations and Results*, Report, 2000.
18. Nilsson, N.J. *Teleo-Reactive Programs and the Triple-Tower Architecture*, Electronic Transactions on Artificial Intelligence 5:99-110 (2001)
19. Ryan, M.R.K. and Pendrith, M.D. An Architecture for Modularity and Re-Use in Reinforcement Learning, *Proceedings 15th International Conference on Machine Learning*, Madison, Wisconsin, 1998.
20. Snedecor, G.W. and Cochran, W.G. Statistical Methods, Iowa State Univ. Press, 1972.
21. Sutton, R. and Barto, A.G. *Reinforcement Learning An Introduction*, MIT Press, 1998.

Implementing an Abstraction Framework for Soft Constraints

Alberto Delgado, Jorge Andrés Pérez, and Camilo Rueda

Pontificia Universidad Javeriana, Cali, Colombia
albertod@puj.edu.co
{japerez, crueda}@atlas.puj.edu.co

Abstract. Soft constraints are flexible schemes for modeling a wide spectrum of problems. A model based on a hierarchy of abstractions of soft constraint problems has been proposed before. We describe an efficient implementation of this scheme aimed at solving real life problems. Our system is integrated into the Mozart language in such a way that user control of the abstraction mechanism is straightforward.We explain how we adapted the theoretical results for our purposes and describe the experiences in this adaptation. We give comparative analysis of our system with respect to an implementation using soft constraints without the abstraction mechanism. Our tests show good performance results for over-constrained problems in real settings.

1 Introduction

A wide variety of problems can be conveniently represented as constraint satisfaction problems (CSP). However, when criteria such as preferences, costs or priorities are involved, more flexible models are needed. In many real world applications criteria of these kind arise naturally, so a good deal of research has been going on for at least one decade in extending CSP to account for them. Researchers have mainly focused on devising a solid theoretical basis for including so-called "softness" in constraint models. Less attention, however, has been payed to the practical side of this line of research.

A mechanism based on problem abstractions for handling "softness" within the concurrent constraint paradigm has been proposed recently ([2, 3]). The idea is to take the concrete (usually hard to solve) CSP and modify it in such a way that an abstract easier problem is obtained. From the solution of the latter valuable clues are obtained for guiding the search of a solution to the former.

To our knowledge, no programming tool has yet been proposed to effectively use the abstraction scheme in real world problems.

We describe a tool written in Mozart ([16]) for abstracting semiring-based constraints. Our abstraction procedures are fully integrated into the Mozart programming language. This approach gives us a number of advantages, including efficiency, correctness (derived from theoretical results), the ability of tackling real-life problems and the possibility of distributing these tools to a large community of users. Our main contribution is thus a tool that supports the abstraction

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 60–75, 2005.

scheme and is fully compatible with Mozart's search model. The tool provides two procedures: one for abstracting a fuzzy CSP into a classical one and another one to bring information from the abstract domain back to the concrete one. This information is very useful to enhance the pruning action of soft constraint propagators, such as those proposed in [6]. We present experimental results on real problems that show the significance of our abstraction procedures for improving efficiency. These results exhibit a good performance of the abstraction procedure, and provide clues about the incidence procedure parameter values have on global performance.

Structure of this Document. The document is organized as follows. In the next section we give a concise account of the theoretical results on semiring-based constraints, including its abstraction scheme. The Mozart model and features are also introduced there. In section 3, our procedures for abstracting soft constraints in Mozart are presented. Analysis and results are described in section 4. In section 5 a revision of related work is given. Finally, a set of concluding remarks as well as some ideas of future work are discussed in section 6.

2 Preliminaries

2.1 Semiring-Based Constraints and Its Abstraction Scheme

Here we briefly summarize the most important definitions and properties of the semiring framework for soft constraints. Theoretical results for abstraction are also outlined. A more complete description of these topics can be found in [1, 2].

A *semiring* is a tuple $(A, +, \times, \mathbf{0}, \mathbf{1})$ where A is a set and $\mathbf{0}, \mathbf{1} \in A$. $+$, the *additive operator* is closed, commutative and associative. Moreover, its unit element is $\mathbf{0}$. $\times$, the *multiplicative operator*, is a closed, associative operation, such that $\mathbf{1}$ is its unit element and $a \times \mathbf{0} = \mathbf{0} = \mathbf{0} \times a$ holds. In addition, $\times$ distributes over $+$. A *c-semiring* is a semiring with some additional properties: $\times$ is commutative, $+$ is idempotent, and $\mathbf{1}$ is its absorbing element. The idempotency of $+$ is needed in order to define a partial ordering $\leq_S$ over the set A, which serves to compare different elements of the semiring. Such a partial order is defined as follows: $a \leq_S b$ iff $a + b = b$.

A *constraint system* is a tuple $CS = \langle S, D, V \rangle$, where S is a semiring, D is a finite set and V is an ordered set of variables. Given a constraint system $CS = \langle S, D, V \rangle$, where $S = (A, +, \times, 0, 1)$, a *constraint* over CS is a pair $\langle def, con \rangle$, where $con \subseteq V$ is called the *type* of the constraint, and $def : D^{|con|} \rightarrow A$ is called the *value* of the constraint. In this way, a *soft constraint problem* (SCSP) P over CS is defined as a pair $P = \langle C, con \rangle$, where C is a set of constraints over CS and con is a subset of V.

Consider any tuple of values t and two sets of variables I and I', with $I' \subseteq I$. $t \downarrow_{I'}^{I}$ denotes the tuple projection of t w.r.t. the variables in I'. Let $c_1 = \langle def_1, con_1 \rangle$ and $c_2 = \langle def_2, con_2 \rangle$ be two constraints over CS. Then, its *combination* $c_1 \otimes c_2$, is the constraint $c' = \langle def', con' \rangle$, where $con' = con_1 \cup con_2$ and $def'(t) = def_1(t \downarrow_{con_1}^{con'}) \times def_2(t \downarrow_{con_2}^{con'})$. Moreover, given the constraint

$c = \langle def, con\rangle$ and a subset w of con, the *projection* of w over c, written $c \Downarrow_w$ is the constraint $\langle def^*, w\rangle$, where $def^*(t^*) = \sum_{\{t | t\downarrow_w^{con} = t^*\}} def(t)$.

Given an SCSP $P = \langle C, con\rangle$ over a constraint system CS, the *solution* of P is a constraint defined as $Sol(P) = (\bigotimes C) \Downarrow_{con}$ where $\bigotimes C$ is the extension of $\times$ to a set of constraints C. Moreover, an *optimal solution* is a pair $\langle t, v\rangle$ such that $def(t) = v$, and there is no t' such that $v < def(t')$. Sometimes it is enough to know the best value associated with the tuples of a solution. This is called the *best level of consistency:* Given an SCSP $P = \langle C, con\rangle$, the best level of consistency for P is defined as $blevel(P) = (\bigotimes C) \Downarrow_\emptyset$. P is said to be consistent if $0 <_S blevel$. In the case where $blevel = \alpha$, P is said to be α-consistent.

By using the ordering $\leq_S$ over the semiring, we can also define a corresponding ordering on constraints with the same type. Consider two constraints c_1,c_2 over a constraint system CS, and assume that $con_1 = con_2$ and $|con_1| = k$. Then $c_1 \sqsubseteq_S c_2$ if and only if, for all k-tuples t of values from D, $def_1(t) \leq_S def_2(t)$. This notion, and the fact that the solution is a constraint. is also useful to define an ordering on problems. Consider two SCSPs $P_1 = \langle C_1, con\rangle$ and $P_2 = \langle C_2, con\rangle$ over CS. Then $P_1 \sqsubseteq_P P_2$ if $Sol(P_1) \sqsubseteq_S Sol(P_2)$.

C-semirings that cast most known variants of CSPs are listed below:

- Classic CSP: $\langle \{false, true\}, \vee, \wedge, false, true\rangle$
- Fuzzy CSP: $\langle \{x \mid x \in [0,1]\}, max, min, 0, 1\rangle$
- Weighted CSP: $\langle \Re^+, min, +, +\infty, 0\rangle$

Abstraction for Semiring-Based Constraints. The idea of abstraction has been adapted for semiring-based constraints in order to relate two versions of an SCSP. This relationship is formally given by a Galois connection.

Let $(\mathcal{C}, \sqsubseteq)$ and $(\mathcal{A}, \leq)$ be two posets (the concrete and the abstract domain). A *Galois connection* $\langle \alpha, \gamma\rangle : (\mathcal{C}, \sqsubseteq) \rightleftharpoons (\mathcal{A}, \leq)$ is a pair of maps $\alpha : \mathcal{C} \to \mathcal{A}$ and $\gamma : \mathcal{A} \to \mathcal{C}$ such that 1) α and γ are monotonic, 2) for each $x \in \mathcal{C}$, $x \sqsubseteq \gamma(\alpha(x))$ and 3) for each $y \in \mathcal{A}$, $\alpha(\gamma(y)) \leq y$. Moreover, a *Galois insertion* (of $\mathcal{A}$ in $\mathcal{C}$) $\langle \alpha, \gamma\rangle : (\mathcal{C}, \sqsubseteq) \rightleftharpoons (\mathcal{A}, \leq)$ is a Galois connection where $\gamma \cdot \alpha = Id_{\mathcal{A}}$. It is possible to establish a relationship between operators in abstract and concrete domains. This relationship is called *local correctness.* Let $f : \mathcal{C}^n \to \mathcal{C}$ be an operator over the concrete domain with an abstract counterpart $\bar{f}$. Then $\bar{f}$ is locally correct w.r.t. f if $\exists x_1, \ldots, x_n \in \mathcal{C}, f(x_1, \ldots, x_n) \sqsubseteq \gamma(\bar{f}(\alpha(x_1), \ldots, \alpha(x_n)))$.

Using this definitions, an abstraction from an SCSP P over a certain semiring S to another SCSP $\bar{P}$ over the semiring $\bar{S}$ can be defined, in such a way that lattices associated to S and $\bar{S}$ are related by a Galois insertion. Specifically, we wish to define an abstraction that preserves the structure of the SCSP. Consider the following concrete SCSP $P = \langle C, con\rangle$ over the semiring S, where

- $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ and
- $C = \{c_0, \ldots, c_n\}$ with $c_i = \langle con_i, def_i\rangle$ and $def_i : D^{|con_i|} \to A$.

Its *abstract counterpart* is defined by an SCSP $\overline{P} = \langle \overline{C}, con\rangle$ over the semiring $\overline{S}$ where:

- $\overline{S} = \langle \overline{A}, \overline{+}, \overline{\times}, \overline{\mathbf{0}}, \overline{\mathbf{1}} \rangle$;
- $\overline{C} = \{\overline{c_0}, \ldots, \overline{c_n}\}$ with $\overline{c_i} = \langle con_i, \overline{def_i} \rangle$ and $\overline{def_i} : D^{|con_i|} \to \overline{A}$;
- if $L = \langle A, \leq \rangle$ is the lattice associated to S and $\overline{L} = \langle \overline{A}, \overline{\leq} \rangle$ the lattice associated to $\overline{S}$, then there is a Galois insertion $\langle \alpha, \gamma \rangle$ such that $\alpha : L \to \overline{L}$;
- $\overline{\times}$ is locally correct w.r.t. $\times$.

Next we list some interesting properties of this abstraction scheme. We will heavily use them on the rest of this paper. In the following we will consider a Galois insertion $\langle \alpha, \gamma \rangle : \langle A, \leq_S \rangle \rightleftharpoons \langle \overline{A}, \leq_{\overline{S}} \rangle$

1. The abstraction of P is the problem $\overline{P} = \alpha(P)$. Applying the concretization function to this abstraction, we obtain the problem $\gamma(\alpha(P))$. These two problems are related by a precise property:

$$P \sqsubseteq_S \gamma(\alpha(P)).$$

   This guarantees that when passing from P to $\gamma(\alpha(P))$ no new inconsistencies are introduced.
2. If applying the abstraction function and then combining gives elements which are in the same ordering as the elements obtained by just combining, the abstraction is said to be *order-preserving*. This fact ensures that an optimal solution in the original problem is also an optimal solution of the abstract one.
3. For any abstraction, it is possible to compute approximate bounds for the valuation of an optimal solution in the concrete domain using an optimal solution of the abstract problem. This is, given an optimal solution of the abstract problem (say t) with valuation $\overline{v}$, we can find an upper and lower bound of an optimal solution for the concrete problem P. Such bounds will be $\gamma(\overline{v})$ and the value of t in P.

For the abstraction that maps fuzzy to classical CSP there are also other interesting results. In this case, the abstraction function is defined by choosing a threshold θ within the interval $[0, 1]$, and mapping all elements in $[0, \theta]$ to 0 and all elements in $(\theta, 1]$ to 1. This abstraction is order-preserving, so we can ensure that the set of optimal solutions of the concrete problem is a subset of the optimal solutions of the abstract one.

1. if $\alpha(P)$ has no solution, problem P has an optimal solution with associated semiring fuzzy value worse than or equal than θ.
2. if P has a solution tuple t with associated semiring level θ, and $\alpha(P)$ has no solution, tuple t is an optimal solution for P.

2.2 Constraint Programming in Mozart

Mozart [16] is a concurrent constraint programming language that provides several functionalities, including support for distributed programming, constraint solving, as well as supporting tools for programming. Many real-life problems have been successfully solved with Mozart (see for instance [7, 8]). It also provides

efficient built in constraints over finite domains of integers as well as convenient mechanisms for creating suitable propagators and new constraint systems [11]. Next we provide a concise introduction to Mozart [13, 10, 15].

Mozart considers basic and non-basic constraints. A *basic constraint* is a logic formula interpreted in some first-order structure. These are chosen so that entailment can be efficiently decided. *Non-basic constraints* are relations built from combination of basic constraints. Basic constraints are kept in a monotonic *store*. Non-basic constraints are enforced by *propagators* [15]. A propagator is a computational agent encapsulating a filter function that deduces consequences (i.e. new basic constraints) of the non-basic constraint. A propagator for a constraint c ceases to exist if c is entailed by the current store or if the conjunction of the current store and c is unsatisfiable. In that case, the propagator for c is said to be *disentailed*, since $\neg c$ is entailed by the current store [10]. Typically, propagators share variables. This causes propagators to trigger each other by writing new basic constraints to the store. This continues until a propagation fixed-point is reached [10]. The order in which the propagators add information to the store does not matter.

Computations in Mozart take place in *computation spaces*. A computation space consists of a set of propagators connected to a store. A space S is said to be *stable*, if no further propagation in S is possible. A stable space S is said to be *failed*, if S contains a propagator that disentails some constraint. A stable space S is *solved*, if S contains no propagators [13]. Moreover, a variable assignment is called a *solution* of a space if it satisfies the constraints in the store and all constraints enforced by propagators.

Constraint propagation is not a complete solution method. To achieve completeness, the space must be *distributed*. Given a stable space S (not failed nor solved), a new constraint c is chosen, and two new spaces must be solved: $S \wedge c$ and $S \wedge \neg c$. It is important to choose c such that both new spaces trigger further constraint propagation. By proceeding in this way we obtain a *search tree*, where each node corresponds to a space and each leaf corresponds to a space that is either solved or failed. Since the alternatives depend on variables of the problem, a finite search tree can be assumed [15].

A *distributor* is an agent implementing a distribution strategy on a sequence $x_1, \ldots x_n$ of variables. When a distribution step is necessary, the strategy selects a yet to be determined variable in the sequence and distributes on this variable (i.e. imposes a new constraint over the selected variable). There are several possibilities for distributing over a variable x. For instance, a *naive* distribution strategy will select the leftmost undetermined variable in the sequence, and adds constraints of the form $x = v$ and $x \neq v$ as alternatives, for some value v.

A Semiring-Based Constraints Solver for Mozart. A soft constraints solver based in fuzzy CSPs has been recently implemented for Mozart (an initial implementation is described in [6]). Although its low-level implementation is fully orthogonal w.r.t. Mozart propagation model, the fuzzy CSP solver constitutes an independent module.

The main feature of the solver is the replacement of the concept of *constraint definition* given in the framework above (which explicitly associates a semiring

value with each tuple) by a notion that is suitable for an inexpensive implementation. Such a notion is defined by three components: a *distance function* wired to each constraint, giving an idea of how wrong a tuple is; a *penalization factor* associated to each constraint, representing the cost that must be payed in the overall valuation when such a constraint is violated, and a *cut level* representing the minimum level of consistency the whole constraint problem must have. The last two notions are user parameters. By the interaction of these notions it is possible to express soft problems in an straightforward way. Valuations associated with each tuple of variables are then computed using the mentioned penalization factor and the distance function. In this way, a small amount of valuation data is stored for a constraint problem while providing propagation algorithms (tailored for each constraint) that discard all tuples valued under the cut level. This is how the cut level influences solving processes for soft constraints.

Currently, the module provides soft versions of several kinds of constraints, including relational operators and arithmetic constraints, using a syntax very similar to the one provided by Mozart's finite domain constraints. Search procedures handling valuations of the solutions are also included in the module. Extending the module (either with constraints and/or search procedures) is straightforward given the constraint propagation interface provided by Mozart [11].

3 Abstracting an SCSP Using Mozart

Finding solutions to a soft constraint problem using conventional constraint programming techniques (i.e. backtracking based ones) turns out to be expensive for several reasons, including the larger search space associated with such a problem, the need of storing and calculating over valuation data and the reduced value pruning action that soft propagation algorithms provide. Clearly, these aspects prevent users from using soft constraints in large or medium size problems. Therefore, finding efficient mechanisms for solving soft constraint problems is crucial for tackling real life problems.

In this scenario, abstraction frameworks constitute a feasible alternative to solve and/or to approximate soft constraint problems in a reasonable amount of time. In particular, the abstraction scheme outlined in section 2.1 provides strong theoretical elements for performing this task. The idea is to process the abstract problem and to extract information from that process, in such a way that the solving process for the concrete problem can be accelerated using information about solutions and/or its approximations. The abstraction scheme can relate several instances of the semiring-based framework such as classical CSP, fuzzy CSP and others. This means that an efficient solver for one of the instances could be used to solve one of the others, under certain assumptions.

In this section we present a Mozart implementation of the abstraction from fuzzy to classical CSPs. This particular abstraction has many interesting properties that can be exploited in an implementation. Moreover, it is possible to take advantage of the efficient classical mechanisms provided by Mozart to implement an expressive soft constraints instance like fuzzy CSPs. Under this idea, there is no need of implementing an additional module or library for including

soft constraints in Mozart programs. Using theoretical results outlined before, we implement procedures to:

- abstract a fuzzy CSP into a classical one (the *alpha function* in a Galois insertion),
- process an abstract problem using an iterative procedure and,
- bring information from the abstract domain to the concrete one (the *gamma function* in a Galois insertion).

In the following we provide a complete description of these procedures, relating the theoretical results described before with the particular features of our Mozart implementation.

3.1 Alpha Function

Alpha function converts a fuzzy CSP into a classical one without affecting the structure of the original problem. By doing so, a mapping between semiring values of the fuzzy CSP (real numbers between 0 and 1) into the two possibles values for the classical CSP (0 or 1) is performed. Those values over a threshold are mapped to 1, while the other values are mapped to 0. In our case, such a threshold is the cut-level of the given problem.

In the semiring formalism every tuple has a semiring valuation associated with it. In our implementation, however, those valuations are computed during execution time using the cut-level of the given problem and the penalization value of each constraint. Consequently, to convert a fuzzy CSP into a classical CSP we modeled it using classical constraints, in such a way that those tuples with valuation over the cut level (of the fuzzy problem) are accepted and all other tuples are rejected. To make this conversion in an automatic way, we defined a classic constraint with a special feature for every fuzzy constraint. Such a feature, so-called *slack value*, is an extra parameter that is computed with the penalization value of every constraint, the cut-level of the whole problem and the maximum valuation possible in the fuzzy semiring (i.e. 1). These constraints with slack values are called *classical counterparts*. In this way, the objective of the alpha function is to compute the slack values for every classical counterpart and to ensure that they accept (and reject) the same values than the fuzzy CSP does.

The intended semantics of the fuzzy constraint must guide the definition of a classical counterpart. In the case of arithmetic/mathematical constraints, a general rule for classical counterparts consists in relaxing the (in)equalities included in them. This can be done by replacing equalities with inequalities and by carefully including slack values in expressions containing inequalities. Note that this unified relaxing criteria is valid for a wide range of constraints, from simple ones like $X \leq Y$ to complicated polynomial constraints. For other types of constraints, e.g. the `all-different` constraint, the relaxation criteria may differ, since several factors may induce or suggest different definitions of classical counterparts for a single constraint. The following example illustrate these ideas.

Example 1. *Consider the fuzzy constraints* $Exp_1 < Exp_2$ *and* $Exp_1 + Exp_2 = Exp_3$*, where each* Exp_i *is an arithmetic expression. The alpha function computes the slack value which, along with the classical counterpart, is used to convert a fuzzy CSP into an equivalent classic CSP. For instance, the classical counterparts for the less than constraint are as follows:*

$$Exp_1 < Exp_2 + S_1 \ \wedge \ Exp_1 - S_1 < Exp_2$$

All tuples accepted by the less than constraint are also accepted by its associated classical counterpart. For instance, let X, Y *be two finite domain variables and* $Exp_1 = X$ *and* $Exp_2 = Y$*, assuming a cut level of 0.8 and a penalization level of 0.05. The slack value (*S_1*) is then equal to 4 (obtained by* $(1.0 - 0.8)/0.05$*). The tuple* $\langle X = 3, Y = 2\rangle$*, valued with 0.9 in the concrete domain, is accepted in the abstract one as both* $3 < 2 + 4$ *and* $3 - 4 < 2$ *hold. On the contrary,* $\langle X = 7, Y = 1\rangle$ *is rejected as both inequalities do not hold (i.e.* $7 \not< 1 + 4$ *and* $7 - 4 \not< 1$*). All tuples accepted by the classical counterpart of a fuzzy constraint are valued with 1. On the other hand, the counterpart for the plus constraint is:*

$$\texttt{abs}((Exp_1 - Exp_2) - Exp_3) \leq S_2.$$

where a possible case could be $Exp_3 = Z$ *(another finite domain variable), and* S_2 *depends on the penalization factor associated with the constraint.*

3.2 Gamma Function

Given a solution to the abstract problem, the gamma function returns its valuation in the concrete domain. This function is used in certain stages of the iterative procedure where the concrete valuation of a solution may improve its performance.

Example 2. *Consider the constraints in the previous example and the tuple* $\langle X = 4, Y = 3, Z = 5\rangle$ *(inconsistent with both of them), assuming a penalization factor of 0.07 for the plus constraint. For this tuple, the gamma function will return an overall fuzzy valuation of* $0.86 = 1.0 - 0.14$*. This penalization is obtained by considering the* ***maximum*** *between the violation cost induced by the plus constraint (*$0.07 * 2 = 0.14$*) and the induced by the less than constraint (*$0.05 * 2 = 0.1$*).*

3.3 An Abstraction Procedure for Soft Constraints in Mozart

Here we describe the implementation of the abstraction scheme proposed in [2]. We explain how alpha and gamma functions fit in our implementation. We use the fact that by finding solutions to the abstracted problem we are finding some possible optimal solutions for the fuzzy problem.

The iterative procedure aims at finding the smallest interval containing the valuation of the best solution to the fuzzy problem. The size of the interval is reduced during the procedure until a desired size is obtained. The interval's bound to be modified depends on the outcome of a search process over the

Algorithm 1. Iterative Algorithm for Abstracting Soft Constraints

```
IterativeSolving := proc (P, Delta, Inter, t, BCut, Option)
  if ValidateInterval(Delta, Inter) == true then
    return BCut
  else
    Spc = StartSpace(P, Inter.low)
    Answ = SearchOneAbstract(Spc)
    if Option == Eager then
      NewInter = EagerMode(Inter, t, Answ, Gamma(Answ))
    else if Option == Binary then
      NewInter = BinaryMode(Inter, t, Answ)
    else if Option == Pessimistic then
      NewInter = PessimisticMode(Inter, t, Answ)
    if Answ == nil then
      IterativeSolving(P, Delta, NewInter, Option, BCut)
    else
      IterativeSolving(P, Delta, NewInter, Option, Inter.low)
```

abstracted problem. A very important feature here is that the cut level used in this search process is given by the lower bound of the current interval. Therefore, the value of this bound is fundamental for overall performance of the iterative procedure. There are three ways of defining the lower bound of the interval (t represents the lowest cut level accepted by the user).

1. *Binary Mode.* If there is a solution in the abstract domain for the interval $[l, u]$, then such an interval becomes $[(l + u)/2, u]$. Otherwise, the interval becomes $[\mathtt{max}(t, 2l - u), l]$.
2. *Eager Mode.* This mode takes into account information from the Gamma function to define the lower bound of the interval. When there is a solution, the new lower bound will be the maximum between the concrete valuation of the found solution (obtained by using Gamma function) and the lower bound given by the Binary mode. Therefore, a lower bound at least as good as the one obtained with the Binary mode is guaranteed.
3. *Pessimistic Mode.* This mode is tailored to those difficult cases when there is no solution or when a solution is very close to 0. Given an interval $[l, u]$, if there is no solution then the interval becomes $[v, l]$ where v is the highest cut level of a solution obtained so far. If such a cut level does not exists, then $v = t$. This mode allows rapidly finding whether there is no solution for the given problem.

Algorithm 1 sketches the iterative procedure described before. It assumes the following input data.

- P, a Mozart procedure asserting a set of soft constraints.
- `Delta`, a real number representing the desired precision.
- Inter, a tuple representing an interval. In the first iteration, this interval is defined as (`low` : `init`, `upp` : 1.0), where `init` is an initial cut level given by the user.

- `t`, the lowest cut level acceptable for the user.
- `BCut`, a real number representing the best cut level found so far.
- Option, a string representing the mode of defining the lower bound of the interval.

A single invocation of the algorithm can be explained as follows. ValidateInterval checks the possibility of reducing the input interval given by the precision `Delta`. If the size of the interval is less than `Delta`, then the best cut level found so far is returned. Otherwise, a computational space is created (function StartSpace). The abstract version of constraints in P is asserted in this space, which takes also the lower bound of the interval as cut level. A search process (function SearchOneAbstract) is then performed over this abstracted problem. The result of this search as well as the lower bound reduction mode are used for determining the new interval. The answer of the search process is used in choosing the recursive call of the algorithm.

Note that the variant of the algorithm is the size of the interval. As this size decreases in each iteration (this is guaranteed by functions Eager and Binary), termination of the algorithm is guaranteed by ValidateInterval. When no solution is found, `nil` is returned.

The iterative procedure provides safe information about the best cut level in the concrete domain. Using that information, a search procedure over the concrete domain is invoked.

Using the Abstraction Procedure in Mozart Programs. The internals of the abstraction procedures are transparent to the user. Constraints are written in the same way as in the concrete (fuzzy CSP) solver. Invocation of the usual search procedures must be replaced by a call to the abstraction procedures.

Abstraction procedures are fully parameterizable. The desired interval size, the initial cut level as well as the mode for defining the lower bound of the interval can be easily provided by the user. Moreover, both abstract and concrete solvers can use graphical Mozart facilities like the Browser [12] and the Explorer [14].

Example 3. *Let us recall the soft constraint problem discussed in examples 1 and 2. Procedure* `Test` *sets up the corresponding abstraction scheme.*

```
proc{Test Sol}
    X Y Z in                          % Declaration of variables
    X::1#6  Y::1#5  Z::3#10           % Domain specification
    {Soft.plus X Y Z 0.07}            % Constraint declaration
    {Soft.less X Y 0.05}
    Sol = sol(x:X y:Y z:Z)            % Defining a Solution Variable
    {FD.distribute ff Sol}            % Distribution strategy (first fail)
end
% Abstraction procedure invokation:
{Soft.abstract Test 0.01 0.6 'Eager'}
```

In this case, the abstraction procedures use a precision of 0.01, with an initial cut level of 0.6 and using the Eager mode for selecting the lower bound.

Note that since the abstraction procedures are completely written in Mozart including them in constraint programs is straightforward. The proposed abstraction scheme could also be implemented in any other constraint programming language.

4 Experimental Results

In this section we illustrate the functionality and features of our abstraction procedures. We study one of the instances of the Radio Link Frequency Assignment Problem (RLFAP) provided by CELAR (the French "Centre d'Electronique de l'Armement") [4]. This problem fits well in our study for several reasons. First, it gives us the opportunity of testing our programming technique with a real life situation. The instance we are dealing with is over-constrained and complex (in terms of the number of variables and constraints). On the other hand, it is a well known benchmark, accessible to anyone interested in solving over-constrained situations in constraint programming and artificial intelligence. The purpose of the presented examples is to illustrate the behavior of the abstraction procedures in over-constrained problems, but not to find their optimal solutions.

Tests in this section were performed on a machine with a 2.4 GHz Xeon Processor running Mozart 1.3.1. All results are the average of 25 runs.

Description of the Problem. The Radio Link Frequency Assignment Problem is a finite domain problem consisting in assign communication channels to radio links from limited spectral resources. In the model, there is a variable for each radio link, and its domain is composed of the available frequencies. Some soft and hard constraints are asserted:

- $x_i = f_j$, asserting that a radio link x_i has a pre-assigned frequency f_j. When the pre-assignation cannot hold, a cost a_i must be payed.
- $|x_i - x_j| > d_{ij}$. This constraint must be imposed when radio links x_i and x_j may interfere together. In case this constraint cannot be satisfied, a cost b_i must be assumed.
- $|x_i - x_j| = \delta_{ij}$. It defines a duplex link. This is a hard constraint asserting that the difference between the distance of the frequency assigned to x_i and the frequency assigned to x_j must be equal to δ_{ij}.

We are interested in studying the behavior and performance of our abstraction procedures in the sixth instance provided by CELAR (simply known as CELAR 6). This instance tries to minimize the sum of violation costs. This optimization criteria is not considered in our tests, although it is possible to include it by giving an additional parameter to the distributor. According to [4], during the process of finding lower bounds for CELAR 6, a set of hard but small sub-instances were extracted. These instances are ideal for benchmarking as they are reasonably hard to solve and can be tackled by current algorithms. They are described in Figure 1 (Left).

Instance	No.of Vars.	No. of Const.	Graph Density
6-0	32	223 (16)	0.4697
6-1	28	314 (14)	0.8306
6-2	32	369 (16)	0.7439
6-3	36	439 (18)	0.6968
6-4	44	499 (22)	0.5274

Starting Level	Time (s)	Valuation
0.6	2.38	`nil`
0.5	173.20	`nil`
0.4	180.73	`nil`
0.3	156.81	0.31
Total time	513.12 s.	

Fig. 1. Left: Instances taken from CELAR 6. The number of hard constraints in each instance is given in brackets. Right: Finding a good cut level with a fuzzy (concrete) solver for instance 6-2

Comparing Abstract and Concrete Solvers. In our first test we compare our abstraction procedures and a soft constraints solver in terms of the required execution time for finding an acceptable cut level. That is, using the soft constraint solver we simulate the abstraction procedures by guessing some values for a cut level and running the soft constraint solver with this level. The purpose is to find the greatest cut level for the problem in the smallest amount of time. In this test, we consider instance 6-2 assuming violation costs of 0.015 for all soft constraints in the problem. Hard constraints were modeled using efficient FD constraints provided by Mozart. Both systems start with a tentative cut level of 0.6. The abstraction procedure considers a precision value of 0.001 using the Eager mode to select the lower bound.

Results are displayed in Figure 1 (Right) and 2. For the soft constraint solver, we picked four different values as tentative cut levels and only found a solution in the last one. Around eight minutes were needed for this. Note that this value is not necessarily optimal.

The results of the abstraction procedures are quite different. Besides the significant time improvements, the abstraction procedure is able to find a lower bound for an optimal solution. This bound is higher than the valuation obtained by the soft constraint solver. The procedure needed nine iterations to achieve a very precise interval. It is interesting to observe that in only one iteration (the eighth one) the value of the best solution found so far was better than the lower bound given by the binary mode.

Choosing Appropriate Initial Cut Levels. Our second test compares the performance of the abstraction procedures for instances 6-0 and 6-4. There is a justification behind this decision. According to [5], with the exception of instance 6-0, every instance 6-i is a sub-instance of 6-$i+1$ and therefore presumably simpler to solve. That is, 6-0 and 6-4 are the only two disjoint instances, and we can consider instance 6-4 as the harder one.

The purpose of this test is threefold. First, a valid concern about our procedures is to determine to what extent a given initial cut level influences the performance of the system. We studied initial cut levels ranging from 0.3 to 0.9. The influence is shown in terms of the number of iterations needed to reach the interval. Second, a particular question regarding RLFAP is the effect of violation

Iteration	Lower Bound	Upper Bound	Solution?
1	0.6	1.0	No
2	0.4	0.6	No
3	0.3	0.4	Yes
4	0.35	0.4	Yes
5	0.375	0.4	No
6	0.3625	0.375	No
7	0.35625	0.3625	No
8	0.353125 *	0.35625	Yes
9	0.355	0.35625	Yes
Total time	5.11 (s)		

Fig. 2. First test: Evolution of the abstraction process for the instance 6-2. Lower bounds are decorated with an '*' when the concrete valuation of the found solution was higher than the binary criteria

Inst.	Best Interval Found (starting with 0.1)	Viol. Cost	Starting Cut Level			
			0.3	0.5	0.7	0.9
6-0	[0.579883, 0.581641]	0.02	268.82 (7)	276.47 (7)	254.99 (8)	307.90 (8)
	[0.685, 0.690625]	0.015	270.29 (7)	229.89 (5)	252.37 (8)	268.79 (9)
	[0.789062, 0.803125]	0.01	275.05 (8)	243.01 (7)	280.41 (6)	243.10 (8)
6-4	[0.14, 0.142188]	0.02	59715.67 (6)	59216.90 (7)	117335.54 (8)	50697.48 (8)
	[0.353125, 0.38125]	0.015	84727.96 (8)	120690.90 (8)	55923.91 (8)	108410.14 (9)
	[0.57, 0.571094]	0.01	85533.80 (8)	83780.74 (7)	81762.58 (8)	49214.30 (8)

Fig. 3. Second Test: Each cell contains the execution time (in milliseconds) and the number of iterations required for finding the optimal cut level. Precision was set to 0.005 for all executions

costs on system performance. We tested three violation costs: 0.01, 0.015 and 0.02. Finally, we wish to obtain a concrete measure of the time performance of our system. In particular, we are interested in studying execution time when both violation costs and the starting cut level vary as described before.

Results for this test are displayed in Figure 3. The second column contains a *reference interval* obtained by running each one of the instances with an initial cut level of 0.1. This shows the effect of violation costs over the interval bounds. Note that the actual interval obtained using the cut levels in Figure 3 can be slightly different from this reference interval.

Our first observation is that instance 6-0 is significantly simpler to solve than instance 6-4, as conjectured in [5]. With respect to the first purpose of the test, by using the reference interval it is possible to infer that *underestimating* the cut level when choosing a starting cut level is a good strategy. This fact is more evident in instance 6-0 with violation costs equal to 0.015. Although in practice the cut level is unknown, it could be an appropriate strategy. Another issue is the interaction between both the number of iterations and the starting cut level with the valuation costs. For executions having high violation cost (e.g. 0.02), as the value of the staring cut level increases, the number of iterations increases

too. This tendency is not very clear in other instances using different valuation costs.

On the relationship between violation costs and time performance, it is not possible to establish a defined behavior from our results. In only one instance (i.e. instance 6-4, with violation cost of 0.01) a clear tendency can be observed. This is a very significant fact, as this is the (theoretically) most expensive instance.

From the results it can be inferred that there is no clear relationship between the number of iterations and time performance. This phenomenon can be explained by the unique features of each problem and by the strong influence slack values have in search processes over the abstracted problem. This also applies for the relationship between violation costs and time performance over problems using the same starting cut level. Although there are cases where the behavior is as expected (e.g. instances 6-4 and 6-0 with starting cut level 0.3), where the average execution time increases as violation costs decrease, most executed problems does not exhibit a defined global behavior.

5 Related Work

The relationship between constraint satisfaction and abstraction formalisms has been previously studied. Most related to ours is [3]. There, an iterative procedure is proposed for solving fuzzy CSPs having a classical solver. Several differences and similarities between our work and [3] can be appreciated:

1. Our implementation and the system reported in [3] are completely different. Our scheme stores just a value for each constraint (the penalization factor), a very inexpensive mechanism compared with the costs of storing and handling valuations associated with each tuple in a soft constraint problem, as done in [3]. In contrast with [3], our proposal considers abstraction as a complete programming technique that *extends* a constraint programming language. Using our procedures, a soft constraint problem can be solved with the same Mozart program, using either the abstract or the concrete solver. Finally, our abstraction procedures are flexible as standard means for extending the classical counterparts are provided. To our knowledge, these capabilities are not available for the system described in [3].
2. With respect to the iterative procedures, options provided by our algorithm are similar to the three versions presented in the algorithm given in [3] (i.e. A1, A2 and A3). However, our modes offer additional features that may improve solution processes. First of all, in our procedures the user is allowed to give both a desired precision for the working interval and a value of the lowest cut level accepted. None of these options is supported in [3], where only a fixed precision of 1/10 is available in the A1 algorithm. As the A2 algorithm in [3], our eager mode also takes into account the valuation of the best solution found to calculate the new lower bound. Nevertheless, we also consider the value of the lower bound given by the binary mode in this calculation, since such a value could be better than the value given by the best solution so far (see the first test in previous section). Finally, our

pessimistic mode improves algorithm A3, since when there is no solution the algorithm continues looking for the best cut level until reaching the lowest cut level given by the user. In contrast, algorithm A3 in [3] stops as soon as no solution is found.

3. Time performance is very similar in both systems. Two important issues should be considered here. First, as said before, the level of precision used in [3] is small compared to ours. Clearly, such a precision influences the number of iterations needed for a problem and thus influences overall performance. Second, the fact that we are dealing with a real-life problem (instead of solving randomly generated problems as in [3]) gives more significance to the time performance of our system.

Another related work ([5]) focuses on finding intervals framing optimum solutions. This work is done in the context of the Valued CSPs and studies how to find upper bounds on the optimum by computing the distance between the value of the best solution found so far and the best lower bound produced so far. In some sense, our work and [5] shares a similar philosophy regarding the role of an optimum, as we intend to approximate it in a very precise way instead of trying to find it.

Finally, in [9] *AbsCon*, an object-oriented tool for solving CSPs using abstraction principles is presented. In that work, the objective is to solve CSP using a classical constraint solver (based on backtracking) possibly in cooperation with a hybrid solver. It considers abstraction as an approximation relation, as opposed to abstraction mappings or Galois connections.

6 Conclusions

In this paper we have used a recently proposed theoretical CSP abstraction framework to construct a complete and robust programming tool for the Mozart programming language. This framework, based on a Galois insertion, is implemented in Mozart in a very clean way, providing straightforward user control.

We analyzed the implementation of alpha and gamma functions in the Mozart search model. Implemented abstraction procedures are highly compatible with an existing module for solving semiring-based constraints in Mozart [6]. This provides a clean interaction between, on the one hand, soft and hard constraints (provided by Mozart), and, on the other hand, our abstraction procedures. In this way, we solve fuzzy problems without implementing a whole new solver. More important, the ideas behind implementation of alpha and gamma functions proposed here for handling soft constraints can be easily applied, without loss of generality, into any programming language providing classical constraints.

Our experimental results show that our abstraction procedures have a very competitive performance for real problems. Abstraction procedures significantly outperformed a fuzzy solver in the search for a good cut level. We have studied the influence of the initial cut level (an input to the abstraction procedure) in the overall process of finding bounds for the best solution in soft constraint problems. By empirical observations, we found that a good strategy is to *underestimate* the

initial cut level. We also have shown how the number of iterations performed by the abstraction procedure has no direct relationship on overall time performance.

In the near future, we plan to provide dynamic slack values inside a unique search tree. This should be more efficient since multiple executions of the abstract solver could be avoided. We also plan to distribute our soft constraints mechanisms (the soft constraints module and the presented abstraction procedures) as a Mozart contribution. This would make our implementations available to anyone interested in this field.

Acknowledgements. We are grateful to Gustavo Gutierrez for useful comments and suggestions on this work. We also would like to thank the anonymous reviewers for their valuable comments for improving this paper.

References

1. S. Bistarelli. *Semirings for Soft Constraint Solving and Programming*. Number 2962 in LNCS. Springer-Verlag, 2004.
2. S. Bistarelli, P. Codognet, and F. Rossi. Abstracting Soft Constraints: Framework, Properties, Examples. *Artificial Intelligence*, 139(2), 2002.
3. S. Bistarelli, F. Rossi, and I. Pilan. Abstracting soft constraints: Some experimental results on fuzzy csps. In *Recent Advances in Constraints*, volume 3010 of *LNCS*, pages 107–123. Springer, 2003.
4. B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J. P. Warners. Radio link frequency assignment. *Constraints: An International Journal*, 4(1), 1999.
5. S. de Givry, G. Verfaillie, and T. Schiex. Bounding the optimum of constraint optimization problems. In *Proc. of CP'97*, volume 1330 of *LNCS*, 1997.
6. A. Delgado, C. Olarte, J. A. Pérez, and C. Rueda. Implementing Semiring-Based Constraints Using Mozart. In *Multiparadigm Programming in Mozart/Oz*, volume 3389 of *LNCS*. Springer-Verlag, 2005.
7. A. Delgado, J. A. Pérez, G. Pabón, R. Jordan, J. F. Díaz, and C. Rueda. An Interactive Tool for the Controlled Execution of an Automated Timetabling Constraint Engine. In *Multiparadigm Programming in Mozart/Oz*, volume 3389 of *LNCS*, pages 322–332. Springer-Verlag, 2005.
8. J. F. Díaz, G. Gutiérrez, C. Olarte, and C. Rueda. CRE2: a CP application for reconfiguring a power distribution network for power losses reduction. In *Proc. of CP'2004*, volume 3258 of *LNCS*. Springer-Verlag, 2004.
9. S. Merchez, C. Lecoutre, and F. Boussemart. Abscon: A prototype to solve csps with abstraction. In *CP 2001*, volume 2239 of *LNCS*. Springer, 2001.
10. T. Müller. *Constraint Propagation in Mozart*. PhD thesis, Universitat des Saarlandes, 2001.
11. T. Muller. The Mozart Constraint Extensions Reference. Available at www.mozart-oz.org, April 2004.
12. K. Popov. The Oz Browser. Available at www.mozart-oz.org, 2004.
13. C. Schulte. *Programming Constraint Services*. PhD thesis, Universitat des Saarlandes, 2001.
14. C. Schulte. Oz Explorer - Visual Constraint Programming Support. Available at www.mozart-oz.org, 2004.
15. C. Schulte and G. Smolka. Finite Domain Constraint Programming in Oz - A Tutorial. Available at www.mozart-oz.org, April 2004.
16. The Mozart Programming Language. http://www.mozart-oz.org.

Transforming and Refining Abstract Constraint Specifications

Alan M. Frisch[1], Brahim Hnich[2], Ian Miguel[3], Barbara M. Smith[2], and Toby Walsh[4]

[1] Dept. of Computer Science, University of York, UK
frisch@cs.york.ac.uk
[2] Cork Constraint Computation Centre, University College Cork, Ireland
{brahim, b.smith}@4c.ucc.ie
[3] School of Computer Science, University of St Andrews, UK
ianm@dcs.st-and.ac.uk
[4] National ICT Australia and Dept. of CS & E, UNSW, Australia
tw@cse.unsw.edu.au

Abstract. To use constraint technology to solve a problem, the solutions to the problem must first be characterised, or *modelled*, by a set of constraints that they must satisfy. A significant part of the modelling process can be characterised as *refinement*, the process of generating a concrete model from an abstract specification of the problem. Expert modellers also identify and perform *transformations* that can dramatically reduce the effort needed to solve the problem by systematic search. Through a case study of modelling a simplified version of the SONET fibre-optic communication problem, this paper examines the processes of refinement and transformation, and especially the interaction between the two.

1 Introduction

Constraint programming is a successful technology for tackling a wide variety of combinatorial problems. To use constraint technology to solve a problem, the solutions to the problem must first be characterised, or *modelled*, by a set of constraints on a set of decision variables that they must satisfy. A significant part of the modelling process can be characterised as *refinement*, the process of generating a concrete model from an *abstract* problem specification. Following [16], and the convention in formal methods, by an abstract specification of a constraint problem, we mean simply a representation in which the details (the modelling decisions) have been abstracted away. Refinement adds these details to produce the concrete model (the modelling decisions are made). There are usually many possible refinements of an abstract problem specification; identifying the effective ones often requires considerable expertise.

Expert modellers also identify and perform *transformations*, which are sometimes complex, that can dramatically reduce the effort needed to solve the problem by systematic search (see, for example, [25]). We use the term *transformation*

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 76–91, 2005.

to refer particularly to operations that change a model or specification but, unlike refinement, do not alter the level of abstraction. Such transformations include adding constraints that are implied by other constraints in the problem, adding constraints that eliminate symmetrical solutions to the problem, removing redundant constraints (i.e., those that yield no extra pruning but add overhead) and replacing constraints with their logical equivalents.

Through a case study of modelling a simplified version of the SONET fibre-optic communication problem [23], this paper examines the processes of refinement and transformation, and especially the interaction between the two. Starting with an abstract specification of the problem, we perform refinements and transformations to produce seven alternative models. These models are *concrete* in that they are similar to the kind that are supported by existing constraint toolkits. We generate the models in an explicit and somewhat systematic way; a sytematic manual exploration of the possible models has proved to be an important first step in our ongoing work towards formalising and automating the modelling process [12].

This case study illustrates the fundamental observation that some transformations operate on a particular (concrete) model of a problem whereas others are model independent. We refer to these two kinds of transformations as *model transformations* and *problem transformations*. Though a problem transformation corresponds to some transformation on a particular model, an advantage of transforming a problem specification is that the benefits of the transformation are inherited by all models. Some transformations can also be performed more easily at the more abstract level of the problem specification.

The case study also shows how a refinement operation can trigger a useful transformation, thus saving the work of searching for it. In particular, we will see a case where a refinement operation that introduces a matrix into a model can easily recognise that the matrix has column symmetry.

Given the ability to generate alternative models, heuristics are needed to guide refinement and transformation towards good models. Towards this goal, we perform an empirical analysis of the generated models to begin to form generalisations about the expected utility of alternative modelling decisions.

2 Specifying the SONET Problem

For illustration, consider the SONET fibre-optic communications problem [22].

> A communications network has client nodes and known levels of demand between pairs of nodes. Traffic can only be routed between two nodes if they are installed, via an add-drop multiplexer (ADM), on the same SONET ring. Each node may be installed on multiple rings and demand between two nodes may be split over several rings. The maximum number of rings available is known. Each ring has a capacity in terms of the volume of traffic and the number of nodes that can be installed on it. Objective: minimise the number of ADMs.

It suffices here to consider a simplified version of the SONET problem, previously considered by Smith [23], in which it is known which node pairs must

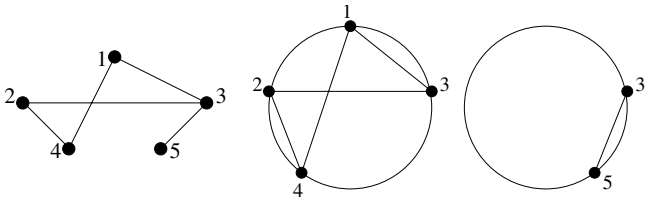


Fig. 1. Demand pairs for and optimal solution of example Simplified SONET instance

communicate, but demand *levels* are ignored. Consider an instance of the SONET problem with 5 nodes and 2 rings, where each ring is able to accommodate 4 nodes. Figure 1 depicts both the demands between nodes and an optimal solution using only 6 ADMs.

2.1 S_A: An Abstract Problem Specification

This section presents an abstract specification of the Simplified SONET problem, which, in subsequent sections, is refined and transformed to produce concrete models. The problem must be specified at a level of abstraction above that at which modelling decisions are made. We use ESSENCE [12], an abstract constraint language whose key feature is that, in addition to the usual atomic variables (variables whose domains comprise atomic elements), it allows non-atomic variables. In doing so, it builds on the facilities available in constraint toolkits such as Ilog Solver and Eclipse, which have supported set variables for several years, ESRA [10], which supports relation variables, $\mathcal{F}$ [17], which supports function variables, and NP-SPEC [7], which supports a variety of variable types, including partitions. However, ESSENCE is unique in that it supports *arbitrarily-nested* variable types, such as set of sets and set of set of tuples.

An instance of Simplified SONET is identified by four parameters: *nnodes*, *nrings* the number of nodes and rings; c, the uniform node capacity per ring; and D, the demand. D is a set of unordered node pairs, $\{n, n'\}$ where n and n' are nodes that must communicate. The decision variable must represent an assignment of nodes to rings; since a node can be assigned to multiple rings, we treat this as a relation, which we call *rings-nodes*, between rings and nodes.

Figure 2 gives S_A, an ESSENCE specification of the Simplified SONET problem[1]. Here the nodes are represented by N, a range of natural numbers. The rings are represented by R, a set comprising *nrings* unnamed elements. The provision of sets of unnamed objects is a unique and important feature of ESSENCE. It facilitates abstraction in specifications by not forcing the elements of a set to be given arbitrary names that are never used. There is no need to name the rings, since individual rings are not mentioned in the specification.

The objective, (1), is to minimise the number of ADMs, represented by the cardinality of *rings-nodes*. The capacity constraint is imposed by (2) and the

[1] Space precludes a full description of ESSENCE. See [12] for details. The simple specification given should be clear from the description.

`given`	*nrings*: `integer`, *nnodes*: `integer`, *c*: `integer`	
`where`	$nrings \geq 0,\ nnodes \geq 0,\ c \geq 0$	
`letting`	N `be integer` $(1..nnodes)$, R `be new type (size` $nrings)$	
`given`	D: `set of set (size` 2) `of` N	
`find`	*rings-nodes*: $R \times N$	
`minimising`	$\lvert rings\text{-}nodes \rvert$	(1)
`such that`	$\forall r \in R.\ \lvert rings\text{-}nodes(r, \_) \} \rvert \leq c$	(2)
	$\forall \{n, n'\} \in D.\ rings\text{-}nodes(\_, n) \cap rings\text{-}nodes(\_, n') \neq \emptyset$	(3)

Fig. 2. Specification of the simplified SONET problem

communication constraint is imposed by (3). To clarify, the expression $\{n, n'\} \in D$ means that two distinct elements are drawn from D and, without loss of generality, one is called n and the other is called n'. Note also that *rings-nodes*$(\_, n)$ is the projection of the *rings-nodes* relation onto $n \in N$, that is $\{r | rings\text{-}nodes(r, n)\}$.

3 Transforming the Abstract Problem Specification

This section presents transformations on the abstract specification of the Simplified SONET problem, S_A. We begin by deriving and adding two implied constraints to the specification.

The first imposes a lower bound on the number of ADMs required for each node. We define the *partner* set of a node n to be $\{n' | \{n, n'\} \in D\}$, each element of which must be related to at least one common ring with n by *rings-nodes*. Once n is installed, the remaining capacity of a ring is $c-1$. Hence, the minimum number of installations of n required to satisfy the communication demand between it and each member of its partner set is $\left\lceil \frac{|\{n' | \{n, n'\} \in D\}|}{c-1} \right\rceil$, to which we refer henceforth as $ADMMin_c(n)$. Observe that all the terms contained in $ADMMin_c(n)$ are parameters; hence, for any given problem instance, it is constant. The implied constraint follows:

$$\forall n \in N. ADMMin_c(n) \leq |rings\text{-}nodes(\_, n)| \tag{4}$$

The second implied constraint imposes a lower bound on the number of 'open' rings, i.e. those rings with at least one node installed. From (4), it is simple to derive a minimum total number of ADMs. Division by the ring capacity c gives the bound $\left\lceil \frac{\sum_{n \in N} ADMMin_c(n)}{c} \right\rceil$, to which we refer henceforth as $RingMin_c$. This is also a constant for any given instance. The implied constraint is:

$$RingMin_c \leq |\{r \in R | rings\text{-}nodes(r, \_) \neq \emptyset\}| \tag{5}$$

We now exploit *dominances*. Given an optimisation problem, a partial assignment a dominates another a' if the utility of the best extension of a is at least as

good as the best extension of a'. We exploit dominances by adding constraints to preclude dominated partial assignments.

First, an assignment where a node n has more installations than the cardinality k of its partner set is dominated by an assignment where n has at most k installations:

$$\forall n \in N. |\{n' | \{n, n'\} \in D\}| \geq |\mathit{rings\text{-}nodes}(\_, n)| \tag{6}$$

Henceforth, we refer to $|\{n' | \{n, n'\} \in D\}|$ as $\mathit{ADMMax}(n)$. It is also constant for any given instance.

Second, an assignment where a node n is installed on a ring that contains no elements of its partner set is dominated by an assignment where n is only installed on rings containing at least one element of its partner set:

$$\begin{aligned} &\forall n \in N, r \in R. \\ &\quad \mathit{rings\text{-}nodes}(r, n) \rightarrow |\{n' \in N | \{n, n'\} \in D \wedge \mathit{rings\text{-}nodes}(r, n')\}| > 0 \end{aligned} \tag{7}$$

Finally, an assignment where the sum of the installations on two non-empty rings is less than or equal to c is dominated by an assignment where the contents of the two rings are merged:

$$\begin{aligned} \forall \{r, r'\} \subseteq R. (\mathit{rings\text{-}nodes}(r, \_) \neq \emptyset) \wedge (\mathit{rings\text{-}nodes}(r', \_) \neq \emptyset) \rightarrow \\ |\mathit{rings\text{-}nodes}(r, \_)| + |\mathit{rings\text{-}nodes}(r', \_)| > c \end{aligned} \tag{8}$$

4 Refining the Simplified SONET Problem

Refining the transformed Simplified SONET specification principally involves replacing the *rings-nodes* relation variable with a structured collection of atomic variables and set variables. If the target language into which we are refining does not support set variables, these could be refined into atomic variables; doing so is not addressed in this paper. We consider three possibilities for refining an arbitrary relation variable, $R : A \times B$:

1. A two-dimensional 0/1 matrix, R_m, indexed by $A \times B$, where $R_m[a, b]$=1 indicates $R(a, b)$, when $a \in A, b \in B$.
2. A one-dimensional matrix of set variables, BtoA_{ms} indexed by A. For each $a \in A$, $\mathit{BtoA}_{ms}[a]$ is $\{b \in B | R(a, b)\}$.
3. A one-dimensional matrix of set variables, AtoB_{ms} indexed by B. For each $b \in B$, $\mathit{AtoB}_{ms}[b]$ is $\{a \in A | R(a, b)\}$.

In the following subsections, we use combinations of these three representations to refine S_A to seven different CSP models, as summarised in Table 1. S_B, S_C and S_D each use one of the three, above-listed representations of a relation variable; the other models each use multiple representations channelled together. Two models, S_B and S_H, closely resemble basic models created by experts in Operations Research [22] and Constraint Programming [23].

Table 1. Simplified SONET: Specification and models

Model	Characteristics
S_A	Sets and Relations
S_B	Matrix
S_C	Ring Set Variables
S_D	Node Set Variables
S_E	Matrix + Ring Set Variables
S_F	Matrix + Node Set Variables
S_G	Ring Set + Node Set Variables
S_H	Matrix + Ring Set Variables + Node Set Variables

4.1 S_B: A Matrix Model

Using rule (1), the *rings-nodes* relation is refined into a two-dimensional matrix of 0/1 variables, $\textit{rings-nodes}_m$, where $\textit{rings-nodes}_m[r, n]$ denotes the element in column r and row n. The matrix needs to be indexed by N and R. Since N is the set $\{1, \ldots, nnodes\}$ it can serve as an index. However, R is an unnamed set, so it cannot serve as an index. We therefore refine R to the set $\{1, \ldots, nrings\}$.

In the ESSENCE statement of the problem there is no way to refer to particular rings, from which it follows that the rings are constrained identically. By naming the rings in S_B we introduced into the model symmetry among rings. In particular, if an assignment is a solution to a Simplified SONET instance, then it is still a solution after we exchange all the nodes installed on any two rings. For example, if installing nodes $\{1, 2\}$ and $\{3, 4\}$ on rings 1 and 2, respectively, is a solution, then so is installing nodes $\{3, 4\}$ and $\{1, 2\}$ on nodes 1 and 2, respectively. Intuitively, the rings are interchangeable. In S_B the rings are the values of the column index of $\textit{rings-nodes}_m$, so the columns of an assignment can be interchanged without affecting whether the assignment is a solution. This is called column symmetry and, in the general case, index symmetry [9].

This discussion illustrates an important observation: refinement can (and often does) introduce symmetry into the model it generates—and it does so in a *systematic* way that can be characterised formally. Indeed, the formal refinement rules presented by Frisch et. al. [12] identify the symmetries that they introduce. The significance of this is that we can avoid the (potentially expensive) process of trying to detect these symmetries in *each* generated model.

Once symmetries are identified, there are several alternative methods that can be used to break them, and thus reduce solution time. One class of methods, called dynamic symmetry breaking (e.g. SBDS [15]), are the symmetry-aware search methods. These search methods take a description of the symmetries and use it to dynamically prune symmetric parts of the search space. Alternatively, the model can be transformed by adding symmetry-breaking constraints that prune some symmetrical assignments from the search space. This is the approach we take here, the advantage of which will become apparent later.

Column symmetry can be dealt with effectively by treating each column as a vector and constraining the columns to be in non-increasing lexicographic order

as the column index increases [9]. Thus, to S_B we add the symmetry breaking constraint:

$$\forall\ 1 \leq r < nrings.\ \mathit{rings\text{-}nodes}_m[r, \_] \geq_{\text{lex}} \mathit{rings\text{-}nodes}_m[r+1, \_] \tag{9}$$

where $\mathit{rings\text{-}nodes}_m[r, \_]$ is column r of $\mathit{rings\text{-}nodes}_m$, and $\geq_{\text{lex}}$ denotes lexicographically greater than or equal to, enforceable by the GACLex algorithm [11].

The nodes N in the Simplified SONET problem are not, in general, interchangeable because they have different demands, as specified by D. However, in certain instances some, or all, of the nodes have identical demands. If a set of nodes has identical demands then the corresponding rows are interchangeable. In Figure 1 n_1 and n_2 have identical demands, so the first two rows of an assignment to $\mathit{rings\text{-}nodes}_m$ can be interchanged without affecting whether the assignment is a solution. Given such a set of interchangeable rows, the symmetry can be broken by constraining them to be in non-increasing lexicographic order as the row index increases. This has been shown to be consistent with the lexicographic ordering constraints that we imposed on the columns [9]. If there are multiple sets of interchangeable rows, each such set can be handled in this way.

The previous paragraph shows how certain symmetries in a model of a particular instance can be handled by adding symmetry-breaking constraints to the model. Our main focus is on building models of problems not instances, so we do not discuss this in detail. However, to handle instance-specific symmetries in a model of a problem, preconditions must be placed on the methods used to break symmetry. As a problem is instantiated into an instance, the preconditions are tested and symmetry is broken among the objects that are symmetrical in the instance.

Now that we have discussed the refinement of *rings-nodes* to a matrix, and the symmetries involved, we continue by refining the constraints and objective function. This requires replacing constraints on *rings-nodes* with constraints on $\mathit{rings\text{-}nodes}_m$. Each column r of $\mathit{rings\text{-}nodes}_m$ corresponds to the characteristic function for the set of nodes installed on ring r (i.e. $\mathit{rings\text{-}nodes}(r, \_)$), and similarly for each row n, so refining (1) and (2) is straightforward:

$$\text{Minimise}(\sum_{r \in R} \sum_{n \in N} \mathit{rings\text{-}nodes}_m[r, n]) \tag{10}$$

$$\forall r \in R.\ \sum_{n \in N} \mathit{rings\text{-}nodes}_m[r, n] \leq c \tag{11}$$

The demand constraint requires the intersection of subsets of N to be non-empty. When using characteristic functions, (3) is easily represented via scalar products, which are the cardinality of the intersection:

$$\forall \{n, n'\} \in D.\text{scalar-product}(\mathit{rings\text{-}nodes}_m[\_, n], \mathit{rings\text{-}nodes}_m[\_, n']) \neq 0 \tag{12}$$

where $\mathit{rings\text{-}nodes}_m[\_, n]$ denotes the nth row of the $\mathit{rings\text{-}nodes}_m$ matrix. S_B is a basic version of that used in [22]. Indeed, matrix models in general are a common pattern in constraint programming [8].

As noted above, each row (or column) of $rings\text{-}nodes_m$ is equivalent to the characteristic function for the projection of *rings-nodes* onto an element of N (or R). A bound on the cardinality of such a projection is easily enforced using a summation on a row or column. Hence (4) and (7) are refined to:

$$\forall n \in N. ADMMin_c(n) \leq \sum_{r \in R} rings\text{-}nodes_m[r, n] \tag{13}$$

$$\forall\, n \in N, r \in R. rings\text{-}nodes_m[r, n] = 1 \rightarrow \sum_{n' | \{n,n'\} \in D} rings\text{-}nodes_m[r, n'] > 0 \tag{14}$$

Constraint (5) places a lower bound on the number of open rings. A ring, r is open if it has at least one ADM installed on it, which in this model means that column r of $rings\text{-}nodes_m$ has a non-zero sum. So, constraint (5) could be implemented by introducing a 0/1 variable for each ring to indicate if it is open. This is cumbersome to impose and it is a weak constraint because it does not force any particular ring to be open.

A much better way of dealing with constraint (5) is obtained by noticing that symmetry-breaking constraint (9) implies that all the open rings are less than[2] the unopen rings. Thus we can impose the constraint that each of the first $RingMin_c$ columns of $ringnodes_m$ has a non-zero sum:

$$\forall\, 1 \leq r \leq RingMin_c. \sum_{n \in N} ringnodes_m[r, n] \neq 0 \tag{15}$$

Observe that this constraint, which is much stronger than merely saying that at least $RingMin_c$ rings are open, can be imposed only because of the symmetry-breaking constraint. In general, the choice between alternative symmetry-breaking constraints should consider the inferred constraints they enable [13]. Also note that this is often a significant advantage to using symmetry-breaking constraints over dynamic symmetry-breaking methods.

Finally, (8) is refined into model S_B straightforwardly, as follows:

$$\begin{array}{l} \forall \{r, r'\} \subseteq R. \\ \quad \sum_{n \in N} rings\text{-}nodes_m[r, n] > 0 \wedge \sum_{n \in N} rings\text{-}nodes_m[r', n] > 0 \\ \quad \rightarrow \sum_{n \in N} (rings\text{-}nodes_m[r, n] + rings\text{-}nodes_m[r', n]) > c \end{array} \tag{16}$$

4.2 S_C: A Set Variable (Rings) Model

Using rule (2), the *rings-nodes* relation is refined into a one-dimensional matrix of set variables, $nodesOnRing_{ms}$, indexed by R such that $nodesOnRing_{ms}[r]$ contains the set of nodes installed on r. As in the previous sub-section, to serve as an index R is refined to the set $\{1, .., nrings\}$.

The objective and ring capacity constraint are easily stated:

$$\text{Minimise}(\sum_{r \in R} |nodesOnRing_{ms}[r]|) \tag{17}$$

$$\forall r \in R. |nodesOnRing_{ms}[r]| \leq c \tag{18}$$

[2] Since each ring is identified by an integer, some rings are "less than" others.

The demand constraint is more difficult to specify. It constrains at least one of the set variables to contain particular pairs of nodes:

$$\forall \{n, n'\} \in D. \sum_{r \in R} (n \in nodesOnRing_{ms}[r] \wedge n' \in nodesOnRing_{ms}[r]) > 0 \quad (19)$$

In the above we have *reified* each conjunction to a 0/1 value and used summation to express the disjunction.

The symmetry among the indices of $nodesOnRing_{ms}$ can be broken cheaply (but only partially) by ordering the cardinalities of the sets:

$$\forall 1 \leq r < nrings.\ |nodesOnRing_{ms}[r]| \geq |nodesOnRing_{ms}[r+1]| \quad (20)$$

Having broken the symmetry in this way, the implied constraint on the minimum number of open rings can be refined simply by disallowing the first $RingMin_c$ elements of $nodesOnRing_{ms}$ from being empty:

$$\forall 1 \leq r \leq RingMin_c. |nodesOnRing[r]| \neq 0 \quad (21)$$

The remaining implied constraint (4) on the minimum number of installations for any node is more awkward since it requires that we check each of the individual rings. Again we use a summation of reified element constraints:

$$\forall n \in N. \sum_{r \in R} n \in nodesOnRing_{ms}[r] \geq ADMMin_c(n) \quad (22)$$

The *ADMMax* constraint to exploit dominance (6) can be stated similarly.

Of the remaining constraints to exploit dominance, $nodesOnRing_{ms}$ facilitates the expression of the content merging constraint (8) most easily:

$$\forall \{r, r'\} \subseteq R. |nodesOnRing_{ms}[r]| > 0 \wedge |nodesOnRing_{ms}[r']| > 0 \rightarrow |nodesOnRing_{ms}[r]| + |nodesOnRing_{ms}[r']| > c \quad (23)$$

Finally, we refine the constraint that specifies a node should only be installed on a ring that contains at least one element of its partner set (7):

$$\forall n \in N, r \in R. n \in nodesOnRing[r] \rightarrow \sum_{n' | \{n,n'\} \in D} (n' \in nodesOnRing[r]) > 0 \quad (24)$$

4.3 S_D: A Set Variable (Nodes) Model

Using rule (3), the *rings-nodes* relation is refined into a one-dimensional matrix of set variables $ringsWithNodes_{ms}$, indexed by N such that $ringsWithNodes_{ms}[n]$ contains the set of rings on which n is installed. Since R is an unnamed set it cannot provide the domain elements for the set variables. Once again, therefore, it is refined to the set $\{1, .., nrings\}$.

Given $ringsWithNode_{ms}$, the objective is refined as follows:

$$\text{Minimise}(\sum_{n \in N} |ringsWithNode_{ms}[n]|) \quad (25)$$

The demand constraints are also easily stated:

$$\forall \{n, n'\} \in D. |ringsWithNode_{ms}[n] \cap ringsWithNode_{ms}[n']| \geq 1 \quad (26)$$

However, since the ring capacity constraints involve all the node constraints we again make use of reification:

$$\forall r \in R. \sum_{n \in N} r \in ringsWithNode_{ms}[n] > 0 \quad (27)$$

The symmetry among the values of $ringsWithNode_{ms}$ can be broken partially by insisting that the first node, n, with a non-empty partner set is installed on the first ring:

$$1 \in ringsWithNode_{ms}[n] \quad (28)$$

Hence, the implied constraint on the minimum number of open rings (5) is refined by ensuring that the first $RingMin_c$ rings appear in at least one of the sets in $ringsWithNode_{ms}$:

$$\forall 1 \leq r \leq RingMin_c. \sum_{n \in N} r \in ringsWithNode[n] \neq 0 \quad (29)$$

The implied constraint on the minimum number of installations per node (4) is easily stated on the node set variables:

$$\forall n \in N. ADMMin_c(n) \leq \sum_{n \in N} |ringsWithNode_{ms}[n]| \quad (30)$$

Again, the $ADMMax$ constraint (6) is specified similarly.

The remaining constraints to exploit dominances, i.e. that a node should only be installed on a ring that contains at least one element of its partner set (7) and the content merging constraint (8), are refined as follows:

$$\forall n \in N, r \in R. r \in ringsWithNode[n] \rightarrow \sum_{n' | \{n,n'\} \in D} (r \in ringsWithNode[n']) > 0 \quad (31)$$

$$\forall \{r, r'\} \subseteq R. (\sum_{n \in N} r \in ringsWithNode[n] > 0) \wedge (\sum_{n \in N} r' \in ringsWithNode[n] > 0) \rightarrow \sum_{n \in N} (r \in ringsWithNode[n] + r' \in ringsWithNode[n]) > c \quad (32)$$

4.4 S_E: A Matrix and Set Variable (Rings) Model

To maintain consistency between $rings\text{-}nodes_m$ and $nodesOnRing_{ms}$, the following channelling constraint is used:

$$\forall r \in R.\ n \in nodesOnRing_{ms}[r] \leftrightarrow rings\text{-}nodes_m[r, n] = 1 \quad (33)$$

The objective can be stated on either $rings\text{-}nodes_m$ (10) or $nodesOnRing_{ms}$ (17). We will explore both alternatives. Symmetry breaking is performed on

rings-nodes$_m$ as in model S_B (9), since this scheme breaks all symmetry whereas the ordering on the cardinalities used in model S_C (20) does not. Also from model S_B we take the demand constraint (12), the *ADMMin* (13) and *ADMMax* constraints, and the constraint that specifies that a node should only be installed on a ring with at least one of its partners (14). From model S_C we take the ring capacity constraints (18), the constraint on the minimum number of open rings (21), and the content merging constraint (23).

4.5 S_F: A Matrix and Set Variable (Nodes) Model

To maintain consistency between *rings-nodes*$_m$ and *ringsWithNode*$_{ms}$, the following channelling constraint is used:

$$\forall n \in N.\ r \in \mathit{ringsWithNode}_{ms}[n] \leftrightarrow \mathit{rings\text{-}nodes}_m[r, n] = 1 \tag{34}$$

The objective can be stated on either *rings-nodes*$_m$ (10) or *ringsWithNode*$_{ms}$ (25). Again, we will explore both alternatives. As in models, S_B and S_E, we perform complete symmetry breaking via *rings-nodes*$_m$ (9). From model S_B we take the ring capacity constraint (11), the constraint on the minimum number of open rings (15), the constraint that a node should only be installed on a ring containing at least one of its partners (14), and the content merging constraint (16). From model S_D we take the demand constraint (26), and the *ADMMin* (30) and *ADMMax* constraints.

4.6 S_G: A Dual Set Variable Model

To maintain consistency between *nodesOnRing*$_{ms}$ and *ringsWithNode*$_{ms}$, the following channelling constraint is used:

$$\forall n \in N, r \in R.n \in \mathit{nodesOnRing}_{ms}[r] \leftrightarrow r \in \mathit{ringsWithNode}_{ms}[n] \tag{35}$$

The objective can be stated on either *nodesOnRing*$_{ms}$ (17) or *ringsWithNode*$_{ms}$ (25). Again, we will explore both alternatives. Symmetry breaking is performed on *nodesOnRing*$_{ms}$, as in model S_C (20). Also from model S_C we take the ring capacity constraint (18), the constraint on the minimum number of open rings (21), the constraint that specifies that a node should only be installed on a ring with at least one of its partners (24), and the content merging constraint (23). From model S_D we take the demand constraint (26), and the *ADMMin* (30) and *ADMMax* constraints.

4.7 S_H: A Matrix and Set Variable (Both Rings and Nodes) Model

Although only two channelling constraints are sufficient to maintain consistency among the 0/1 matrix and the two matrices of set variables, we use the three channelling constraints from models S_E (33), S_F (34) and S_G (35). The objective can be stated easily on any of the three models. We will explore all three alternatives. Symmetry is broken completely on *rings-nodes*$_{ms}$ as described in model S_B (9). Also from model S_B we take the constraint that a node should only be installed on a ring containing at least one of its partners (24). From model S_C we

take the ring capacity constraint (18), the minimum number of open rings (21), and the content merging constraint (23). From model S_D we take the demand constraint (26), and the *ADMMin* (30) and *ADMMax* constraints.

5 Model Selection

As we have shown in introducing each of the models S_B to S_H, constraints may be more or less difficult to express, depending on the variables included in the model. However, which model is best in solving time, given some standard constraint solver, is hard to determine. In some cases, it is possible to show that one model is stronger than another, irrespective of certain aspects of the solution procedure. Recently, for example, alternative models of permutation and injection problems have been studied in the context of a range of constraint propagation algorithms [18]. In many cases, however, empirical tests are needed to develop guidelines for making informed model choices. In this section, we contribute to this goal of pattern elicitation by performing an empirical analysis of our models of the Simplified SONET problem. Despite the small scale of this study, the trends are strong and immediately apparent.

There are a number of issues to consider in designing our experiment. First, introducing new variables can introduce a choice of search variables. For instance, in model S_E, we can search either on the matrix variables $rings\text{-}nodes_m$ or on the ring set variables $nodesOnRing_{ms}$. Second, having chosen the search variables, we need to decide the order in which to assign the variables (either statically or dynamically). It is well known that the choice of variable ordering can dramatically affect the search effort required to solve a CSP. However, we can only compare the performance of the models presented to a limited extent. For instance, we could compare models S_B, S_E and S_F using the matrix variables as search variables and the same variable ordering in each case. This would show whether being able to express some of the constraints more easily using set variables has any effect on performance. The results would not, however, necessarily reflect the best known performance for these models, still less what the best performance for each model might be with the ideal ordering heuristic.

Each of our models is described by a triple $\langle BasicModel, BranchingStrategy, ObjectiveExpression\rangle$. $BasicModel \in \{B, C, D, E, F, G, H\}$ corresponds to the models S_B - S_H. We considered $BranchingStrategy \in \{M, N, R\}$ where M stands for using the matrix variables as search variables, N for using the node set variables, and R for the using the ring set variables. Finally, we considered $ObjectiveExpression \in \{M, N, R\}$, where M stands for expressing the objective function using the matrix variables (10), N stands for using the node set variables (17), and R for the ring set variables (25). We tested the 24 consistent combinations of these three choices on 10 instances (from [23]) using a 750Mhz 128Mb Pentium III and Ilog Solver 5.3 (Windows version).

Tables 2 and 3 present the number of choice points and time taken within a 160 seconds time limit. For brevity, given a subset of models that are identical apart from the objective function, we only show the results for the model with

Table 2. Experimental results on 10 instances of Simplified SONET (choices)

Model	s2ring1	s2ring2	s2ring3	s2ring4	s2ring5	s2ring6	s2ring7	s2ring8	s2ring9	s2ring10
$\langle B, M, M\rangle$	25411	27063	20032	7938	24097	9625	8460	10001	41849	9428
$\langle C, R, R\rangle$	>514K	>514K	>468K	>492K	>518K	>495K	441K	>512K	>506K	>513K
$\langle D, N, N\rangle$	12744	374983	63892	16771	48955	400641	25311	78181	239662	15680
$\langle E, M, R\rangle$	7971	5421	4765	1583	8601	2491	1394	2597	15912	3761
$\langle E, R, R\rangle$	68395	199656	87765	36343	134385	140771	15301	103778	225852	41991
$\langle F, M, N\rangle$	**112**	**165**	**73**	**21**	**356**	**193**	**39**	**188**	**1136**	**70**
$\langle F, N, N\rangle$	1407	17303	3804	2525	2758	27920	2702	6218	4329	1387
$\langle G, R, N\rangle$	378K	>675K	>636K	>634K	>663K	>614K	256K	>688K	>668K	>461K
$\langle G, N, N\rangle$	20629	>546K	67994	56995	98936	>642K	51673	78982	229565	41031
$\langle H, M, N\rangle$	**112**	**165**	**73**	**21**	**356**	**193**	**39**	**188**	**1136**	**70**
$\langle H, R, N\rangle$	12876	217440	23569	20472	84782	87275	10559	145915	52622	7572
$\langle H, N, N\rangle$	1407	17303	3804	2525	2758	27920	2702	6218	4329	1387

Table 3. Experimental results on 10 instances of Simplified SONET (time)

Model	s2ring1	s2ring2	s2ring3	s2ring4	s2ring5	s2ring6	s2ring7	s2ring8	s2ring9	s2ring10
$\langle B, M, M\rangle$	2.49	2.78	2.29	0.86	2.41	1.03	0.97	1.08	4.2	1.05
$\langle C, R, R\rangle$	>160	>160	>160	>160	>160	>160	132.44	>160	>160	>160
$\langle D, N, N\rangle$	6.7	158.37	32.44	7.64	24.32	160.02	13.5	33.26	115.96	8
$\langle E, M, R\rangle$	1.22	0.88	0.79	0.26	1.44	0.42	0.25	0.47	2.7	0.65
$\langle E, R, R\rangle$	14.76	39.43	18.74	7.47	27.05	29.11	3.22	18.95	44.76	9.47
$\langle F, M, N\rangle$	**0.03**	**0.04**	**0.03**	**0.02**	**0.08**	**0.04**	**0.02**	**0.04**	**0.19**	**0.03**
$\langle F, N, N\rangle$	0.25	2.29	0.6	0.39	0.43	3.45	0.44	0.82	0.71	0.24
$\langle G, R, N\rangle$	92.13	>160	>160	>160	>160	>160	60.95	>160	>160	>160
$\langle G, N, N\rangle$	6.47	>160	22.37	15.86	24.34	>160	15.3	18.03	79.66	11.75
$\langle H, M, N\rangle$	0.05	0.07	0.05	0.03	0.13	0.07	0.03	0.07	0.38	0.04
$\langle H, R, N\rangle$	3.06	46.86	5.81	4.29	16.43	18.6	2.33	27.54	11.87	1.94
$\langle H, N, N\rangle$	0.47	4.32	1.23	0.75	0.73	6.2	0.87	1.45	1.37	0.41

the most effective expression of the objective. For instance, we do not show the results for model $\langle E, M, M\rangle$, since $\langle E, M, R\rangle$ is consistently more effective.

This filter immediately reveals a general observation: in these tests it was always more effective to express the objective via the cardinality of the set variables. This is because of an interaction with the *RingMin* constraint (5) in the case of the ring set variables, and with the *ADMMin* constraint (4) in the case of the node set variables. Consider $rings\text{-}nodes_m$ for a small instance:

$$\begin{array}{c} \\ node_1 \\ node_2 \\ node_3 \end{array} \begin{array}{c} \begin{array}{ccc} ring_1 & ring_2 & ring_3 \end{array} \\ \left(\begin{array}{ccc} 0/1 & 0/1 & 0/1 \\ 0/1 & 0/1 & 0/1 \\ 0/1 & 0/1 & 0/1 \end{array}\right) \end{array}$$

Assume that all nodes need to be installed at least once, and that all three rings must be open. These constraints can be imposed as sums on the rows and columns of $rings\text{-}nodes_m$. Propagating these constraints results in no domain pruning initially. Consider the search for a solution with less than 3 installations. Since all elements of $rings\text{-}nodes_m$ can still be set to 0, search is necessary to determine that this is not possible.

Consider now expressing the *RingMin* constraint and the objective on the $nodesOnRing_{ms}$ matrix. The lower bound on the cardinality of each ring is one.

Hence, the sum of the cardinalities is at least 3 and the search fails immediately. Expressing the *ADMMin* constraint and the objective on $ringsWithNode_{ms}$ gives a similar result. The key observation is that the bounds directly tighten the domain of a variable (the hidden cardinality variable associated with each set variable). Since these same variables are used to express the objective, any tightening of the bounds has a direct effect on the bound on the objective. This is not the case for the sum constraints on $rings\text{-}nodes_m$. Since the *ADMMin* constraint gives a tighter bound on the cardinality variables than *RingMin*, this also explains why expressing the objective on $ringsWithNode_{ms}$ is the most effective choice in these experiments.

A second observation is that it is most effective to branch on $rings\text{-}nodes_m$. This is probably due to the fact that assigning a single 0/1 variable is less of a commitment than assigning a whole set at once. Hence, the culprit at a dead end is more readily apparent. These two observations together explain the performance of $\langle F, M, N\rangle$ as the best model. Model $\langle H, M, N\rangle$ explores the same search tree, but incurs an overhead for maintaining $nodesOnRing_{ms}$.

6 Related Work in Modelling and Transformation

Several recent efforts focus on automating refinement. Hnich [17] shows how to automatically refine specifications in $\mathcal{F}$ and Frisch et. al. [12] show how to refine specifications in Essence. Both of these refinement systems generate a set of alternative models, including models with multiple, channelled representations, but neither provides a mechanism for choosing among the alternatives. The initial implementation of Relational ESRA, which is under development, will refine specifications to a single constraint model in which relation variables are always refined to 0/1 matrices (as in S_B) [10].

Our work is also motivated by experience with the CGrass (Constraint Generation And Symmetry-breaking [14]) system. CGrass automatically transforms constraint models of problem *instances* in order to make them easier to solve. However, since CGrass transforms individual instances, much effort is repeated if one wants to solve multiple instances of a problem. Furthermore, the instance specifications that CGrass transforms are non-schematic; instead of using universal quantifiers to implicitly state a set of constraints, the set is explicitly stated. For some problem instances this results in very large specifications, which, in turn, require many applications of the transforation rules. This is why we focus on schematic problem specifications in this paper.

There are several other methods to aid in constraint modelling, which we briefly survey. Laurière [19] introduced a modelling language called ALICE to formally state a problem. The language is characterised by the use of sets, set operators, Cartesian product of sets, vectors, matrices, graphs and paths, constants, and functions. The language NP-SPEC is a logic-based executable specification language [7, 6], which allows the user to specify problems by using metapredicates (*subset*, *partition*, *permutation*, and *intfunc*). REFINE is a functional language for specifying global search problems for a program synthesizer [24]. The RE-

FINE language augments a functional programming language with three type constructors, namely *set*, *sequence*, and *map*, as well as their operations.

Tsang *et al.* had two projects related to ours. The adaptive constraint satisfaction project [1, 2, 3] aimed at systematically mapping problems, in a dynamic manner, to algorithms and heuristics. The computer-aided constraint programming project [4, 21] aimed at building a system that encapsulates the entire process of applying CP technology to problems.

7 Conclusions

We have considered the transformation of constraint satisfaction problems and shown that we can and should transform problems at various levels of abstraction. Refinement is a process of progressively moving to more concrete models; mechanisms for dealing with some common modelling problems, such as symmetry, can be embedded into refinement rules.

The Simplified SONET problem illustrates how integrating transformation and refinement could work in general: an abstract problem specification in the ESSENCE language was transformed by adding implied and other constraints. The result was refined into seven alternative models. In addition, we showed how the refinement process could trigger further useful transformations: in this case, breaking the symmetries that it introduces into the model.

Our immediate goal is to formalise fully the transformations we use. Furthermore, we wish to combine theoretical analysis with the lessons learnt from empirical analyses, like the one performed on the Simplified SONET problem herein, to evaluate models statically, and therefore be more selective about the models produced during refinement.

Acknowledgements. We thank Sherali and Smith for providing sample instances. Brahim Hnich is supported by Science Foundation Ireland. Ian Miguel is supported by a UK-Royal Academy of Engineering/EPSRC Research Fellowship. This work was done while the fourth author was employed at the University of Huddersfield.

References

1. A. Abbas and E.P.K. Tsang. Toward a general language for the specification of constraint satisfaction problems. *Proc. Constraint Programming, Artificial Intelligence and Operations Research (CP-AI-OR) Wshop*, 2001.
2. J.E. Borrett. *Formulation selection for constraint satisfaction problem: a heuristic approach.* PhD Thesis, Dept. of Computer Science, University of Essex, UK, 1998.
3. J.E. Borrett and E.P.K. Tsang. A context for constraint satisfaction problem formulation selection. *Constraints*, 6(4):299-327, 2001.
4. R. Bradwell, J. Ford, P. Mills, E.P.K. Tsang, and R. Williams. An overview of the CACP project: modelling and solving constraint satisfaction/optimisation problems with minimal expert intervention. *Proc. Wshop on Analysis and Visualization of Constraint Programs & Solvers*, 2000.

5. A. Bundy. A Science of Reasoning. J-L. Lassez and G, Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, 178–198, 1991.
6. M. Cadoli and A. Schaerf. Compiling program specifications into SAT. *Proc. ESOP'01*. LNCS 2028. Springer-Verlag, 2001.
7. M. Cadoli, L. Palopoli, A. Schaerf, and D. Vasile. NP-SPEC: An executable specification language for solving all problems in NP. *Proc. PADL'99*, pp. 16–30. LNCS 1551. Springer-Verlag, 1999.
8. P. Flener, A.M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, T. Walsh. Matrix Modelling: Exploiting Common Patterns in Constraint Programming *Proc. Int. Wshop on Reformulating CSPs*, 2002.
9. P. Flener, A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking Row and Column Symmetries in Matrix Models. *Proc. 8th Int. Conf. on Principles & Practice of CP* LNCS 2470, 2002.
10. P. Flener, J. Pearson, and M. Agren. Introducing ESRA, a relational language for modelling combinatorial problems. *LOPSTR'03: Revised Selected Papers*, LNCS 3018, 2004.
11. A.M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, T. Walsh. Global Constraints for Lexicographic Orderings. *Proc. 8th Int. Conf. on Principles & Practice of CP* LNCS 2470, 2002.
12. A.M. Frisch, C. Jefferson, B. Martínez-Hernández, I. Miguel. *The Rules of Constraint Modelling. Proc. 19th Int. Joint Conf. on AI*, 2005.
13. A.M. Frisch, C. Jefferson, I. Miguel. *Symmetry Breaking as a Prelude to Implied Constraints: A Constraint Modelling Pattern. Proc. 16th Euro. Conf. on AI*, 171-175, 2004.
14. A.M. Frisch, I. Miguel, and T. Walsh. Cgrass: A System for Transforming Constraint Satisfaction Problems. *Proc. Joint Wshop of the ERCIM/CologNet Area on CP & CLP*, LNCS 2627, 23–26, 2002.
15. I. P. Gent and B. M. Smith. Symmetry Breaking During Search in Constraint Programming. *Proc. European Conf. on AI*, 599–603, 2000.
16. F. Giunchiglia, T. Walsh. A Theory of Abstraction. *Artificial Intelligence* 56(2–3), 323-390, 1992
17. B. Hnich. *Function Variables for Constraint Programming*. PhD Thesis, University of Uppsala, 2003.
18. B. Hnich, B.M. Smith and T. Walsh. Models of Permutation and Injection Problems. *Journal of Artificial Intelligence Research*, 21, 2004.
19. J-L. Lauriere. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, 10(1):29-127, 1978.
20. A.K. Mackworth. Constraint Satisfaction Problems. *Encyclopedia of AI*, 285–293, 1992.
21. P. Mills, E.P.K. Tsang, R. Williams, J. Ford, and J. Borrett. EaCL 1.5: An Easy abstract Constraint optimisation Programming Language, TR CSM-324, University of Essex, 1999.
22. H.D. Sherali and J.C. Smith. Improving Discrete Model Representations via Symmetry Considerations. *Management Science* 47, 1396–1407, 2001.
23. B. M. Smith Symmetry and Search in a Network Design Problem. *Proc. 2nd Int. Conf. on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, LNCS 3524, 336-350, 2005.
24. D.R. Smith. The structure and design of global search algorithms. *TR KES.U.87.12*, Kestrel Institute, 1988.
25. B. M. Smith, K. Stergiou, and T. Walsh. Modelling the Golomb Ruler Problem. *Proc. IJCAI-99 Wshop on Non-Binary Constraints*. Int. Joint Conf. on AI, 1999.

Learning Regular Expressions from Noisy Sequences

Ugo Galassi and Attilio Giordana

Dipartimento di Informatica, Università Amedeo Avogadro,
Via Bellini 25G, 15100 - Alessandria, Italy
{galassi, attilio}@mfn.unipmn.it

Abstract. The presence of long gaps dramatically increases the difficulty of detecting and characterizing complex events hidden in long sequences. In order to cope with this problem, a learning algorithm based on an abstraction mechanism is proposed: it can infer the general model of complex events from a set of learning sequences. Events are described by means of regular expressions, and the abstraction mechanism is based on the substitution property of regular languages. The induction algorithm proceeds bottom-up, progressively coarsening the sequence granularity, letting correlations between subsequences, separated by long gaps, naturally emerge. Two abstraction operators are defined. The first one detects, and abstracts into non-terminal symbols, regular expressions not containing iterative constructs. The second one detects and abstracts iterated subsequences. By interleaving the two operators, regular expressions in general form may be inferred. Both operators are based on string alignment algorithms taken from bio-informatics. A restricted form of the algorithm has already been outlined in previous papers, where the emphasis was on applications. Here, the algorithm, in an extended version, is described and analyzed into details.

1 Introduction

Very long discrete sequences are found in many challenging applications of data mining, ranging from DNA analysis to user profiling, and anti-intrusion systems. In most cases this kind of sequences are characterized by sparseness, i.e., short consecutive chains of atomic events (episodes) are interleaved with gaps, where irrelevant facts, or facts related to spurious activities, may occur. We define a partially ordered group of interrelated episodes a *complex event* (CE).

This paper addresses the task of discovering CEs in discrete sequences. The task is made more difficult by assuming the presence of noise, making CEs harder to recognize. Episodes are represented as strings of symbols, being a symbol the label assigned to an atomic event. Moreover, it is assumed that noise can be modeled as insertion, deletion and substitution errors, according to a common practice followed in Pattern Recognition.

Here, regular expressions, extended with attributes [9], are proposed to describe the structure of CEs. Attributes are used to set constraints on atomic

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 92–106, 2005.

events. Therefore, the problem of discovering CE's structure is turned into the problem of learning regular expressions from sequences containing gaps and noise. The problem of inferring regular grammars from data has been previously investigated by many authors with approaches ranging from computational learning theory [1, 19, 16, 17, 4] to neural networks [6], syntactic pattern recognition [10, 18], and probabilistic automata [8]. Nevertheless, the problem considered here does not match immediately any one of the problems solved by the mentioned approaches. In fact, the task is more complex, because the sentences of the language to learn are hidden inside sequences containing a possibly large amount of irrelevant knowledge, which must be discarded.

In a previous approach [2] a Hierarchical Hidden Markov Model [7, 22, 14, 21] has been proposed to describe CEs. Here, we prefer to distinguish the problem of learning structural properties of a CE from the problem of detecting it in presence of noise and other irrelevant facts. Therefore, an extension of the algorithm by Botta et al. [2] is presented, which is more powerful and is no more bound to HHMMs.

By exploiting properties inherent to regular expressions, an abstraction mechanism has been defined: it allows an event to be seen at different levels of granularity depending on the needs. Such a mechanism is exploited by the learning algorithm, which automatically infers the event descriptions from a database of sequences. An important novelty, with respect to previous works, is a method for detecting and learning recurrent structures inside an event, in presence of noise.

In this paper, the learning algorithm is described in details and an evaluation on artificial data is provided.

2 Learning by Abstraction

The main difficulty in discovering and modeling CEs hidden inside long sequences is due to the presence of long gaps, filled by irrelevant facts, between episodes belonging to a CE. On the one hand, statistical correlations among distant episodes are difficult to detect. On the other hand, the complexity of the *mining* algorithm increases with the length of the portion of sequence to be searched to detect such kind of correlations. The strategy proposed here to cope with such kind of problems is based on an abstraction mechanism.

In AI, abstraction has been proposed by several authors with different acceptions (see [20] for an introduction). The acception, adopted here, relays on the property of regular expressions of being closed under substitution [13]: by replacing a subexpression with a new symbol, an abstract expression is obtained. As previously mentioned, CEs are described by means of regular expressions extended with attributes. By applying the substitution property, a CE can be abstracted, or de-abstracted.

The idea will be further clarified describing the scheme of the algorithm used for discovering CEs hidden in a set, $\mathcal{LS}$, of learning sequences. The algorithm starts bottom-up to construct an abstraction hierarchy, layer after layer. The basic activity at each step consists in identifying episodes occurring with a relevant

frequency in $\mathcal{LS}$: every episode is characterized by a regular expression $\mathcal{R}$. Then, the detected episodes are *named* by associating a new symbol to each one of them, and episode names become the alphabet for describing $\mathcal{LS}$ at the next abstraction level. Afterwards, every sequence in $\mathcal{LS}$ is abstracted (rewritten) by replacing every episode instance occurring in it with the corresponding episode name. Subsequences of consecutive atomic events, which have not been included in any episode, are replaced with a symbol denoting a *gap*. As it will be described in the next sections, gaps between episodes are considered as a special kind of episode.

In the new sequences obtained from the abstraction step, episodes, previously separated by subsequences of irrelevant facts, may become consecutive, only separated by one gap symbol. Then, at the next abstraction step, correlations at a wider range can be detected by repeating the same procedure described so far, while the complexity of the algorithm remains affordable.

Important aspects to consider, in order to correctly detect statistical correlations between consecutive atomic events, are the event duration and the distance from one another, which could be required to satisfy specific constraints. As an example, one may be willing to accept a correlation between two events A and B, when B frequently occurs few days after A, but one may want to reject a correlation if the distance of B from A randomly ranges from one day to one year.

The attributes extending regular expressions have principally the function of preserving the information about duration and distance between events through the abstraction process. Every atomic event E is denoted by a name (symbol) and by an attribute l_E reporting the length (duration) of E on the unabstracted sequence. When an episode is abstracted into a new atomic event at the higher level, the length of this last is set to the length of the episode. In the same way, gaps are denoted by a symbol, and have a length set to the distance between the two neighbouring episodes. As it will be described in the following, the event description language allows constraints on duration of an event to be specified. Therefore, to set constraints on the distance between two events is sufficient to set constraints on the gap in between.

This solution, of using gap symbols to fill spaces between non adjacent atomic events, allows for any discrete sequence to be transformed into a *string* of symbols. The important benefit is that a large set of string processing algorithms can be immediately exploited.

3 Regular Expressions

The standard formalism for regular expressions [13] is adopted for describing episodes and CEs. Regular language syntax contains meta-symbols for denoting disjunction and iteration. Disjunction is denoted by the symbol "|". For instance, the construct $a(c|d)a$ denotes a sequence of three symbols, where the first and the third are "a", and the second may be "c" or "d". Parentheses are used to enclose subexpressions. The special symbol ϵ denotes the null event and is used to model omission. For instance, expression $a(c|d|\epsilon)a$ entails that also the sentence aa, is a possible event instance. Repetition is denoted by a superscript on a symbol, or

on a subexpression, which indicates how many consecutive times it occurs. As an example, expression a^3b^2 is a compact form for denoting the sequence "aaabb".

In principle, regular expressions can also describe infinite sentences. The classical notation for handling infinity consists in using symbol "$\star$" as a superscript to expressions. Here, infinity is not allowed. Instead, the regular language notation is slightly extended to allow for nondeterminate iterations, where the number of repetitions may range inside a bounded interval. For instance, expression $ab^{3,9}$ denotes a sequence whose first element is "a" followed by a number of "b" ranging between 3 to 9.

Constraints on the event/gap length may be set by annotating symbols in regular expressions. Annotation must be included inside square brackets, following the symbol denoting an atomic event. For instance $a[n]$ means that the length l_a of a must be n ($l_a = n$), whereas $a[n, m]$ means that the length of a must range between n and m ($n \leq l_a \leq m$). A legal example of annotation can be as in the following:

$$a[3, 5]^3 b[4, 8]^2 \tag{1}$$

Informally, expression (1) specifies that the duration of any event of type a must be in the interval $[3, 5]$ and the duration of any event of type b must be in the interval $[4, 8]$. Gaps are named and annotated as atomic events. However, given the semantics of gaps, iteration has no meaning for them; then, gap names cannot have an exponent.

4 String Alignment and Flexible Matching

A key role in the abstraction process is played by the *approximate matching* of strings and of regular expressions, which, in turn, is based on *string alignment*. String alignment has been deeply investigated in Bio-informatics and a wide collection of effective algorithms are available for doing it[5, 12]. Here some basic concepts, necessary to make the paper self-consistent, will be recalled; the interested reader can find in[5, 12] an exhaustive introduction to the topic.

Definition 1. *Given two strings s_1 and s_2, let s'_1 and s'_2 be two strings obtained from s_1 and s_2, respectively, by inserting an arbitrary number of spaces such that the atomic events in the two strings can be put in a one-to-one correspondence. The pair $A(s_1, s_2) = \langle s'_1, s'_2 \rangle$, is said a global alignment between s_1 and s_2.*

From global alignment, local alignment and multi-alignment can be defined.

Definition 2. *Any global alignment between a pair of substrings r_1 and r_2 extracted from two strings s_1 and s_2, respectively, is said a local alignment $LA(s_1, s_2)$, between s_1 and s_2.*

Definition 3. *Given a set S of strings, a multi-alignment $MA(S)$ on S is a set S' of strings, where every string $s \in S$ generates a corresponding string $s' \in S'$ by inserting a proper number of spaces, and every pair of strings $\langle s'_1, s'_2 \rangle$ is a global alignment $A(s_1, s_2)$ of the corresponding strings s_1, s_2 in set the S.*

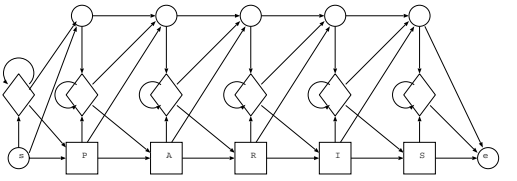


Fig. 1. Profile HMM obtained from the string "PARIS". Square nodes represent *match states*, circles represent *null emission states* and diamonds represents *insertion states*. Transitions, from one state to another, and emissions are governed by probability distributions not shown in the figure. States labeled by s and e are the initial and final state, respectively

It is immediate to verify that, for a pair of strings s_1 and s_2, many alignments exist [1]. However, the interest is for alignments maximizing (or minimizing) an assigned scoring function[2]. A typical scoring function is string similarity [12], which can be stated in the following general form:

$$\int(s_1, s_2) = \sum_{i=1}^{n} \int(s_1'(i), s_2'(i)) \tag{2}$$

being n the length of the alignment $\langle s_1', s_2' \rangle$, and $\int(.,.)$ a scoring function, which depends upon the symbol pairs, which have been aligned.

An alternative to (2) for aligning strings and estimating similarity is based on a special kind of Hidden Markov Model (HMM) called *profile HMM* (see [5] for an introduction). The fundamental difference between profile HMM and (2) is that for the former the scoring function is stated in terms of a mixture model defining a probability distribution. Then the similarity between two strings s_1 and s_2, or between a string and a template, is defined as the probability that s_2 be obtained from s_1 as the result of a stochastic sequence of insertions, deletions and substitutions. The structure of a profile HMM is described in Figure 1. It contains three types of states: *match states* where the emission corresponds to the expected nominal symbol, *null emission states* modeling deletion errors, and *insertion states* modeling insertion errors, where the emission is chosen among a set of possible symbols. Such a structure can be obtained by compilation from a string, as well as from a regular expression. In the case of Figure 1 the HMM has been obtained from the string "PARIS".

In the framework of Dynamic Programming, the problem of finding an alignment maximizing a similarity function is solvable with complexity of $O(nm)$ being n and m the length of s_1 and s_2, respectively. Nevertheless, approximate solutions can be found in linear time[12]. On the contrary, the problem of find-

[1] If no restriction is set on the possible number of inserted spaces, the number of possible alignments is infinite.

[2] As *approximate/flexible matching* between two strings, or between a string and a regular expression, is intended the problem of finding the optimal alignment with respect to an assigned scoring function.

ing an optimal multi-alignment is exponential in the cardinality $|S|$ of set S. Therefore, only approximate solutions can be used when S is large.

The concept of similarity and alignment between strings is easy to extend to the concept of alignment between a string and a regular expression. A regular expression $\mathcal{R}$ is equivalent to a set of strings that can be derived from it. Therefore the optimal alignment between $\mathcal{R}$ and a string s, with respect to an assigned similarity function, is the best alignment among all possible alignments between s and anyone of the strings derivable from $\mathcal{R}$. In the general case, the complexity for finding such an alignment is $O(nm)$ being m the length of $\mathcal{R}$ and n the length of s [15].

A similar extension holds in the HMM framework, where regular expressions can be translated into HMMs. However, such translation requires the target HMM to be augmented in two ways: (a) in order to deal with the presence of insertion and deletion errors, extra states must be explicitly added; (b) in order to model specific probability distributions, cycles in regular expressions need to be unrolled into a feed-forward graph, where only self-loops are allowed. A description of the problem and of the related methodologies can be found in [5, 2, 11].

A last point to discuss is how constraints, set in regular expressions on event lengths, intervene in the matching procedure. Dealing with such kind of constraints requires only minor changes in the algorithms searching for an optimal alignment: symbols in the input string not matching the constraints will be considered as insertion errors that do not match any symbols. Consequently, the impact on the final alignment will depend upon the specific scoring function. In a similar way, considering iterated subexpressions, iterations in excess (defect), with respect to the bounds set in the exponent, will be considered as insertion (deletion) errors.

5 The Learning Algorithm

The main learning algorithm includes a basic cycle, activated bottom-up, in which a new abstraction layer is constructed, and a refinement cycle, which can be called top-down one or more times in order to refine the episode descriptions (see Figure 2). Both cycles are based on two abstraction operators, ω_S and ω_I, which are used to infer the structure of regular expressions. Operator ω_S constructs regular expressions non containing iterative constructs, whereas ω_I explicitly aims at discovering and abstracting iterative constructs. By interleaving the two operators, an abstraction hierarchy is obtained, from which regular expressions in general form are obtained.

5.1 ω_S Operator

The ω_S operator takes in input a set S of similar substrings, detected using a local alignment algorithm, and constructs an abstract atomic event defined as a pair $\langle \mathcal{R}, E \rangle$ being $\mathcal{R}$ a regular expression generalizing the episode instances contained in S, and E is the abstract event associated to $\mathcal{R}$. The restriction is

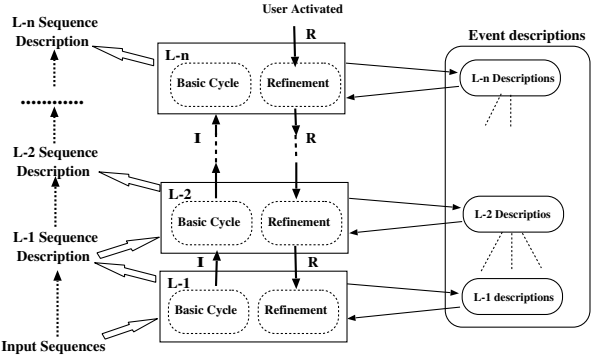


Fig. 2. The main learning algorithm structure. Every layer produces a more abstract description of the input sequences

that items in $\mathcal{R}$ may be only symbols, or disjunction of symbols. Therefore, no iterative constructs are considered.

The core of ω_S is the construction of the multi-alignment of all strings in the set S; the similarity measure and the alignment procedure are parameters, which can be assigned according to the needs. The semantics of ω_S consists in the following three step algorithm:

Algorithm ω_S

1. Construct the multi-alignment $MA(S)$ for strings in S.
2. Construct the match graph $MG(S)$.
3. Transform $MG(S)$ into an equivalent regular expression.

The multi-alignment $MA(S)$ is a table whose columns contain the symbols put in correspondence by the alignment algorithm. Therefore, the second step aims at eliminating noise from episode descriptions preserving possible multi-modalities. Symbols, occurring in a same column more frequently than expected if they would be due to random noise, are considered *match symbols* and will be included in the regular expression generated in the third step. Match symbols are associated to the nodes of a directed graph $MG(S)$. The edges of $MG(S)$ are defined according to the following rule: if there is at least one row in $MA(S)$ where a match symbol x follows a match symbol y, immediately or after one or more spaces, a link from x to y is set in $MG(S)$.

Graph $MG(S)$ is transformed into a regular expression in Step 3. As there are many possible way for doing it, it is not relevant to describe the algorithm into details. In this phase, constraints on the event length (see Section 3) are also learned. We remember that an atomic event E, in a regular expression, can be annotated as $E[n, m]$, being n and m the extremes of the interval in which the event length l_E is accepted. The values for n and m are estimated from the lengths of the instances of E aligned in a same column in $MA(S)$. In this phase, constraints, given a-priori as background knowledge, can also be taken into account.

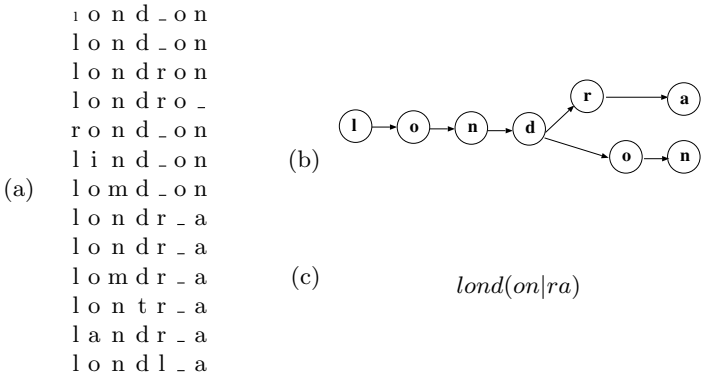


Fig. 3. Example of non-iterative expression obtained from the string set {london, londra, lomdra, lontra, londro, londron, rondon, lindon, london, lomdon, landra, londra, londla}. (a) Corresponding multi-alignment. (b) Retained alternatives. (c) Final regular expression

The algorithm is illustrated through an example in Figure 3, where a regular expression describing a dimorphic occurrence of the word *london*[3] in the Italian language is extracted from a set of words affected by typos.

5.2 ω_I Operator

Operator ω_I is complementary to ω_S, and explicitly searches for contiguous repetitions of a same substring inside a given string s. This is done by computing the self-correlation of the string similarity function. In fact, repeated substring instances are expected to be very similar each other (identical in absence of noise). Then, periods in self-similarity function locate where repetitions of a same substring occur. Let W_i and w_j denote a reference window and a sliding window (of equal size) on s beginning in position i and j, respectively. Let n be the length of s minus the length of W_i. Let, moreover, SC be a triangular matrix of size $n^2/2$; the notation $SC(i, j)$ will indicate the i, j element of SC. The basic self-correlation algorithm is the following:

Similarity self-correlation

1. Set i = 1
2. For j ranging from i to n evaluate $SC(i, j) = \int(W_i, w_j)$ between the substrings selected by W_i and w_j, respectively.
3. Set i = i+1
4. If i is smaller than n goto step 2, otherwise continue.

[3] Names of foreign towns may occur in an Italian text both in their original orthographic form, or in Italian translation. In this case London is translated into "Londra".

5. Detect chains of maxima on SC, where the maximum value is close to the maximum possible similarity value between two substrings W_i, w_j. A substring r of s, laying in between two consecutive maxima, is an iterated substring.
6. For every different iterated substring r construct a new hypothesis for an iterated episode.

The complexity of the algorithm is $O(n^2/2)$. However, it is easy to make the algorithm more efficient: when string similarity is close to zero, the windows can slide much faster than one position at-a-time. The contour plot of the SC matrix of two different strings is reported in Figure 4 (for the sake of clarity, the entire square matrix has been computed).

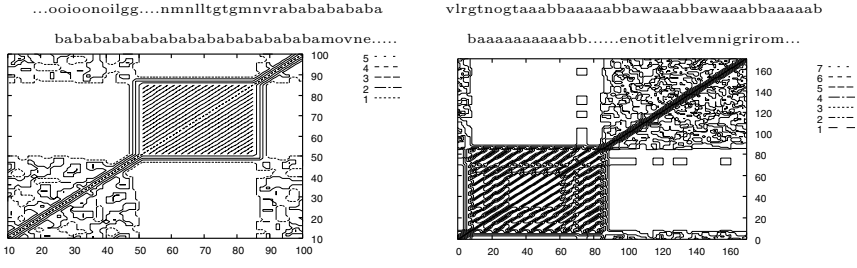


Fig. 4. Examples of similarity self-correlation patterns. (a) The repeated subsequence is "ab". (b) The repeated subsequence is "aaaaabb". In both sequences noise has been added. The rectangular patterns clearly indicate the region where the iterated subsequences are located

5.3 Basic Learning Cycle

The basic learning cycle consists of four major steps:

1. Non-iterative episode detection. Episodes consisting of non-iterated substrings are detected and abstracted by applying operator ω_S.
2. Iterative episode detection. Episodes consisting of an iterated substring are detected and abstracted by applying operator ω_I.
3. Model construction. When necessary, an HMM is constructed for every abstracted episode.
4. Sequence abstraction. The input sequences are rewritten using as new alphabet the names of the abstract episodes.

Non-iterative Episode Detection. The core mechanism is represented by the abstraction operators ω_S. However, some preprocessing is required before applying the operator. In fact, ω_S takes in input a set S of strings that is constructed by applying a local alignment algorithm LA (see Section 4) followed by a clustering algorithm. More specifically, LA is repeatedly applied to a set of sequence pairs randomly sampled from the learning set $\mathcal{LS}$ and produces in

output pairs of subsequences that exhibit strong similarity. It is expected that a frequent episode occurs in many pairs of sequences with minor differences from one instance to another. Then, episodes deriving from a same regular expression are easy to detect by using a clustering algorithm, which groups together most similar subsequences. The specific algorithm used for this step is not very much critical, because the refinement cycle allows possible errors to be recovered, as it will be explained later on. The currently used algorithm is an incremental variant of classical k-Means.

Finally, ω_S is applied to every cluster S obtained in this way, constructing a corresponding abstract event.

Iterative Episode Detection. In principle, the procedure described above is able to discover an iterated episode when the number of iterations is very similar in all sequences where the episode occurs. On the contrary, it does not work properly when the number of iterations is significantly different from one sequence to another, because the multi-alignment step fails. This problem is solved by operator ω_I, which is applied to a set of sequences sampled from $\mathcal{LS}$. All iterated episodes found in this way are collected into a set I. Afterwards, episodes characterized by an identical (or very similar) iterated substring are generalized to a unique abstract episode description: a common iterated subsequence is chosen, and the iteration limits are set in order to include all found instances. The abstract events constructed in this way are then added to the ones generated by operator ω_S.

Model Construction. This step is accomplished only if an approximate matching based on HMM has been required and consists in constructing an HMM for every abstract event E characterized in the previous steps. Every expression $\mathcal{R}_E$ is converted into a HMM λ_E, and the sets of substrings, used to learn the regular expression describing the abstract events, are used to estimated the parameters of λ_E. The details of the algorithm can be found in [2, 11].

Sequence Abstraction. Every sequence s in $\mathcal{LS}$ is rewritten into an abstracted sequence s' according to the following algorithm: s is scanned left-to-right searching for instances of episodes detected and abstracted in the previous steps. The presence of an episode E is decided by matching the corresponding regular expression $\mathcal{R}_E$ to s. Every time an instance is found, the *name* of E is appended to s'. However, conflicting interpretations of a same subsequence may exist. Conflict resolution is delayed to a second swept and, initially, a lattice is generated, containing all plausible hypotheses for episode instances. Afterward, lattices are processed extracting from each one the maximum scoring sequence, which includes the best scored hypotheses compatible with the given constraints. The default constraint is that hypotheses must not overlap. In the case a string similarity function of type (2) is used to match regular expressions, the score assigned to episode hypotheses is the value computed by the similarity function. Otherwise, if a matching based on HMM is used, the score of an event E is the

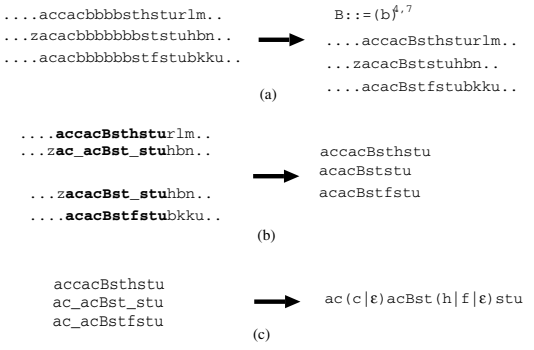


Fig. 5. Basic learning cycle example. (a) Iterated symbols are detected and replaced with the name of the corresponding regular expression. (b) Local alignments are detected and similar substrings are clustered together. (c) From the multiple alignment of elements in a same cluster a regular expression is obtained

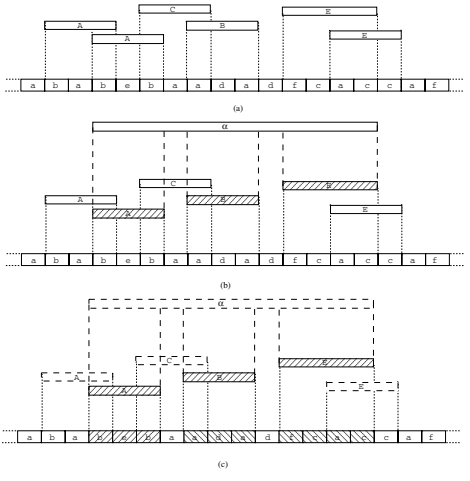


Fig. 6. The refinement step. (a) Episode lattice. (b) Some hypothesized events in (a) are not considered for a new episode. (c) Only the retained instances are used to re-train the episode model

probability assigned by the model λ_E. Portions of the string s not abstracted by any episode are abstracted as gaps and represented by a gap symbol.

The major steps of the basic cycle are illustrated through an example in Figure 5.

5.4 Refinement Cycle

The refinement cycle may be activated at the abstraction layer L_i every time new episodes are detected and modeled at a level higher than i. The reason for doing

it is illustrated in Figure 6. When an episode E is hypothesized and characterized at an abstraction level L_i, the context, i.e, the presence of other episodes before or after E, is not considered. Nevertheless, the context is considered later on when the episodes of layer L_i are linked together into an episode at level L_{i+1}. This means that some instances of E may be not included in any higher level episode and will be considered spurious. Nevertheless, such instances were included in the cluster used to build up the regular expression describing E. In the refinement step, the regular expression describing E are re-learned using only the instances that have been retained.

As episode instances are detected using one of the approximate matching algorithms described in in Section 4, the outcome of the refining cycle heavily depends on it. Therefore, using a similarity function or an HMM can produce quite different results.

6 Discussion and Evaluation

The learning algorithm described in previous sections has been implemented in two different versions, where the major difference is in the flexible matching algorithm. In the first version, which is an extension of the work described in [3], matching is based on a string similarity function. In the second version [2], string similarity is still used in the basic learning cycle, whereas, in the refinement cycle, regular expressions are translated into HMMs. More specifically, the cascade of regular expressions generated by the abstraction mechanism leads to a Hierarchical HMM [7], which is trained using the classical EM algorithm. The two different versions have been tested both on artificial data and on real world problems. Artificial data are a suitable tool to evaluate learning algorithms, because they can be constructed on purpose to put in evidence both strong and weak points. A first benchmark between the two algorithm versions has been run using a suite of datasets consisting of artificially generated sequences of symbols. The structure for all datasets is similar, and consists of sequences of words selected from natural language and interleaved with gaps filled by randomly chosen characters. Moreover, noise is added by randomly replacing an assigned percentage of characters with other characters randomly extracted from the alphabet. The challenge for the algorithms is to reconstruct the regular expression corresponding to the sequence of words hidden in the data. Every dataset contains 100 sequences and the global length of the sequences ranges from 60 to 140 characters.

The difficulty of the task has been controlled by varying three parameters: (a) the number of words ($5 \leq w \leq 8$) in the regular expression; (b) the word length ($5 \leq L \leq 8$); (c) the noise level ($N \in \{0\%, 5\%, 10\%, 15\%\}$. For every triple $< w, L, N >$, 10 different datasets have been generated for a total of 640 learning problems.

The results comparing the two algorithms are summarized in Table 1. The error rate (averaged on 10 problems) is evaluated as the edit distance (i.e., the minimum number of corrections) between the regular expression learned by the

Table 1. Performances of the two algorithm versions obtained on artificial datasets. The sequence length ranges from 60 to 140 characters. The CPU time for solving a problem ranges from 42 to 83 seconds on a Pentium IV 2.4 Ghz

		Similarity function				HMM			
		Noise Level				Noise Level			
w	L	0%	5%	10 %	15%	0%	5%	10 %	15%
5	5	0.00	0.00	0.00	0.00	0.04	0.04	0.04	0.04
5	6	0.01	0.01	0.02	0.01	0.03	0.03	0.03	0.03
5	7	0.00	0.03	0.02	0.02	0.00	0.00	0.02	0.00
5	8	0.01	0.03	0.03	0.04	0.00	0.00	0.00	0.00
6	5	0.00	0.02	0.03	0.02	0.10	0.06	0.00	0.03
6	6	0.02	0.04	0.01	0.01	0.05	0.00	0.00	0.00
6	7	0.03	0.02	0.02	0.04	0.02	0.00	0.00	0.00
6	8	0.05	0.06	0.04	0.04	0.00	0.00	0.04	0.00
7	5	0.02	0.01	0.01	0.01	0.02	0.05	0.01	0.10
7	6	0.02	0.00	0.01	0.01	0.04	0.02	0.05	0.04
7	7	0.04	0.02	0.02	0.04	0.00	0.00	0.02	0.05
7	8	0.04	0.02	0.02	0.04	0.01	0.00	0.09	0.09
8	5	0.03	0.01	0.01	0.01	0.00	0.00	0.01	0.00
8	6	0.01	0.00	0.01	0.01	0.03	0.06	0.06	0.14
8	7	0.07	0.03	0.04	0.05	0.00	0.00	0.00	0.00
8	8	0.12	0.06	0.05	0.07	0.01	0.00	0.00	0.00
Avg.		0.026	0.022	0.021	0.026	0.021	0.016	0.023	0.026

algorithms and the original one hidden in the data. When an entire word is missed, the corresponding error is set equal to its length. Experiments in Table 1, reporting an error rate much higher than the others, have missed words. In all cases, the learning cycle has been iterated twice, as explained in Section 5. Cells in the last row reports the average of the error rates in the corresponding column.

From the results of Table 1 the two algorithm versions show performances substantially similar, and show good robustness with respect to the presence of noise. However, from a more detailed analysis of the table, some differences emerge. Flexible matching based on HMM is slightly superior to similarity based one when the word length increases. Moreover, it shows a less stable behavior: either the performances are very good, with a zero error rate, or the error rate is quite high, because one or more words have been missed in some learning problem.

It is worth noting that both versions exhibit better performances when episodes to detect are longer. This is easy to explain, because long-range regularities are easier to distinguish from noise than short-range ones. Then, in general, longer CEs are expected to be easier to learn than shorter ones.

As previously mentioned, the learning algorithm has been applied also to a real world problem where the goal was to learn the profile of a user typing on a keyboard. The description of the task and the obtained results are described in [11]. In this case, the algorithm version based on HMM clearly dominated the other version. The reasons are to be searched in the greater complexity of the data set and on the nature of the data, which better fit an HMM. In fact, the similarity function (2) does not take into account the location of the errors in the episodes whereas HMM does. Therefore, it is not possible to establish a-priori which version would perform better, if the nature of the data is not known.

7 Conclusion

An algorithm for discovering complex events in noisy sequences has been presented, where complex events are described as regular expressions. The main contribution of the paper consists in organizing and generalizing in a unique framework different methods developed in the past. Moreover, for the first time the learning algorithm architecture, which is based on an abstraction mechanism, is described into details.

Currently, two versions of the algorithm exist, which have been developed by integrating large part of previous work: one makes use of string similarity in order to match regular expressions on noisy data; the other one translates regular expressions into a hierarchical Hidden Markov Model [7]. Both versions exhibit good performances on artificial data, whereas the second one was superior in solving a non trivial user profiling problem, where it was required to learn the model for a user editing a text.

In both versions, the algorithm has been easy to apply and didn't require special tuning on the problem. This means that the method is robust and suitable for applications in real domains.

However, the evaluation is not yet complete. On the one hand, the ability of the algorithm at learning regular expressions where iteration is deeply involved has been only partially tested, and the results obtained up to now are not yet conclusive. On the other hand, a theoretical analysis of the convergency properties of the algorithm is in progress.

Acknowledgments

The present work has been supported by the FIRB Project: WebMinds.

References

1. D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.
2. M. Botta, U. Galassi, and A.Giordana. Learning complex and sparse events in long sequences. In *Proceedings of the European Conference on Artificial Intelligence, ECAI-04*, Valencia, Spain, August 2004.
3. M. Botta, A. Giordana, and P. Terenziani. Discovering complex events in long sequences. In *Proceedings of the "Workshop on learning in temporal sequences", Machine Learning Conference*, Sidney, Australia, July 2002.
4. F. Denis. Learning regular languages from simple positive examples. *Machine Learning*, 44(1/2):37–66, 2001.
5. R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis.* Cambridge University Press, 1998.
6. J. L. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7:195–225, 1991.
7. S. Fine, Y Singer, and N. Tishby. The hierarchical hidden markov model: Analysis and applications. *Machine Learning*, 32:41–62, 1998.

8. P. Frasconi and Y. Bengio. An em approach to grammatical inference: iputo/output hmms. In *Proceedings of International Conference on Pattern Recognition, ICPR-94*, 1994.
9. K. S. Fu. *Syntactic pattern recognition and applications*. Prentice Hall, 1982.
10. K.S. FU and T.L. Booth. Grammatical inference: Introduction and survey (part 1). *IEEE Transaction on System, Men and Cybernetics*, 5:85–111, 1975.
11. U. Galassi, A. Giordana, and D. Mendola. Learning user profiles from traces. *Technical report TR-INF-2005-04-02-UNIPMN*, 2005.
12. D. Gussfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.
13. J.E. Hopcroft and J.D. Ullman. *Formal languages and their relation to automata*. Addison-Wesley, 1969.
14. K. Murphy and M. Paskin. Linear time inference in hierarchical hmms. In *Advances in Neural Information Processing Systems (NIPS-01)*, volume 14, 2001.
15. E.W. Myers and W. Miller. Approximate matching of regular expressions. *Bulletin of Mathematical Biology*, 51(2):5 – 37, 1989.
16. R. G. Parekh and V. G. Honavar. Learning DFA from simple examples. In *Proceedings of the 8th International Workshop on Algorithmic Learning Theory (ALT'97), Lecture Notes in Artificial Intelligence*, volume 1316, pages 116 –131, Sendai, Japan, 1997. Springer.
17. Rajesh Parekh, Codrin Nichitiu, and Vasant Honavar. A polynomial time incremental algorithm for learning DFA. *Lecture Notes in Computer Science*, 1433: 37–50, 1998.
18. P. Garca and E. Vidal. Inference of k-testable languages in the strict sense and applications to syntactic pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9):920–925, 1990.
19. S. Porat and J. Feldman. Learning automata from ordered examples. *Machine Learning*, 7:109–138, 1991.
20. L. Saitta, editor. *The abstraction paths, Special issue of the Philosophical Transactions of Royal Society, Series B*. 2003.
21. M. Skounakis, M. Craven, and S. Ray. Hierarchical hidden markov models for information extraction. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence IJCAI-03*, pages x–x. Morgan Kaufmann, 2003.
22. L. Xie, S. Chang, A. Divakaran, and H. Sun. *Learning hierarchical hidden Markov models for video structure discovery*, volume Tech. Rep. 2002-006. ADVENT Group, Columbia University, December 2002.

From Factorial and Hierarchical HMM to Bayesian Network: A Representation Change Algorithm

Sylvain Gelly, Nicolas Bredeche, and Michèle Sebag

Equipe Inference & Apprentissage - Projet TAO (INRIA futurs),
Laboratoire de Recherche en Informatique,
Université Paris-Sud, 91405 Orsay Cedex, France
{gelly, bredeche, sebag}@lri.fr
http://tao.lri.fr

Abstract. Factorial Hierarchical Hidden Markov Models (FHHMM) provides a powerful way to endow an autonomous mobile robot with efficient map-building and map-navigation behaviors. However, the inference mechanism in FHHMM has seldom been studied. In this paper, we suggest an algorithm that transforms a FHHMM into a Bayesian Network in order to be able to perform inference. As a matter of fact, inference in Bayesian Network is a well-known mechanism and this representation formalism provides a well grounded theoretical background that may help us to achieve our goal. The algorithm we present can handle two problems arising in such a representation change: (1) the cost due to taking into account multiple dependencies between variables (e.g. compute $P(Y|X_1, X_2, ..., X_n)$), and (2) the removal of the directed cycles that may be present in the source graph. Finally, we show that our model is able to learn faster than a classical Bayesian network based representation when few (or unreliable) data is available, which is a key feature when it comes to mobile robotics.

1 Introduction

Many works in mobile robotics rely on probabilistic models such as POMDP or HMM[1], etc.) to build a map of an environment [2, 1, 7, 4, 5]. Indeed, the properties of these models are particularly relevant in the context of robotics, as well as extensions of these models. Firstly, the problem of knowledge generalization can partly be solved if we consider a hierarchical model (encode a given place at sereral granularities) [6]. Secondly, taking into account the invariants can also be achieved if we consider a model that implements a factorization operator (e.g. a given place location should be perceived with no considerations for the actual

[1] In the following of the article, we deal with HMM rather than with POMDP. The particularity of the latter being that they explicitly take into account action, which is not a key issue for the inference problem at hand.

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 107–120, 2005.

orientation of the robot) [4]. However these two extensions have been well studied separately, it is quite difficult to endow a HMM-based model with these two simultaneously. As far as we know, there exists no efficient inference algorithm that can deal with such a model.

In this paper, we present an approach to perform inference within a Factorial and Hierarchical HMM (i.e. FHHMM[2]). Our approach relies on an algorithm that performs a representation change from FHHMM to the Bayesian Network representation formalism. The choice of the Bayesian Network formalism is motivated by the strong theoritical fundations and the efficient algorithms that exists in it.

However, several difficulties arise with such a representation change because of the structural differences between the two formalisms and their intrinsic properties. In particular, we identify two main problems that must be taken into account during this process:

- There exists multiple dependencies in the FHHMM. These implies an exponential growth of the number of parameters to learn, which is a challenging problem when dealing with a small set of example (this is an intrinsic property in mobile robotics) ;
- There exists directed cycles in the conditional dependencies between the variables of a FHHMM. It is well known that directed cycles are not allowed within a Bayesian network (we should note however that these dependencies are a problem only between variables at a same time step (see section 2)).

In the following section, we present the HMM formalism and the factorial and hierarchical extensions. Then, we describe the inference problem in the case of FHHMM. Section 3 and 4 presents our approach along with the representation change algorithm. Lastly, section 5 presents two experiments which confront the resulting model and classical Bayesian networks for a learning task. We conclude this paper with a discussion about the interesting properties shown by our model as well as the compromise we made so as to be able to learn from few data, which is often the case of a mobile robot building a map of its environment.

2 Problem Setting

2.1 Hierarchical and Factorial HMM

Known limitations with HMM, and more generaly with markov models, are concerned with scaling, taking into account independent phenomena and the difficulty to generalize. However, there exists several extensions to solve this problem. In the following, we focus our attention on hierarchical HMM [7, 5] and factorial HMM [3][3].

[2] We use this abrevation in the following of the article.

[3] These extensions have been used separately (with POMDPs) for map-building by a robot [5, 4].

On the one hand, the hierarchical extension allows to reduce the number of links between the states of an HMM, and then reduce the algorithmic complexity of learning as well as improving the acurracy. On the other hand, the factorial extension makes it possible to explain observations with several causes rather than only one. In this case, the goal is to turn the $P(Y|X)$ of HMM into $P(Y|X^1, X^2, ..., X^n)$. The X^i are hidden variables and can be dealt with separatly. Thus, the $P(X^i_{t+1}|X^i_t)$ are different for each i.

2.2 Conditional Dependencies and Sparse Data

Let's begin by introducing the following definitions:

- A static dependency denotes the conditional dependency between two variables at the same time step. It is important to notice that the problem of directed cycles arise only from this kind of dependencies.
- A dynamic dependency is defined as a conditional dependency for two (e.g. classical HMM) or several variables between two time steps (e.g. factorial HMM).

Classic and hierarchical HMM contain only dynamic dependencies. However, static dependencies can be found in the case of factorial HMM when conditional dependencies are to be created between some variables.

In the scope of this paper, we consider a special kind of HMM, where the dependencies type may be *a priori* undefined. As a matter of fact, dynamic and static dependencies are both expressed as conditional dependencies within the Bayesian network formalism.

2.3 Problem Issues

Since we consider an HMM that implements both the factorial and hierarchical extensions along with undefined dependencies, we face the problem of finding a fitted inference algorithm. As a matter of fact, there do not exists any such algorithms for this kind of model. This is the first issue: how to perform inference in such a model.

Another important issue is that due to the original motivation (i.e. mobile robotics), we have to consider the case where there is few data to learn from. Indeed, the sample process is supposed to be controlled by the robot's behavior and the environment, which usually gives few and biaised examples. Hence, we state that a good property of our model would be to favor the learning speed even at the cost of a (reasonable) loss in accuracy.

3 Representation Change: From FHHMM to Bayesian Networks

3.1 Constrained Representation Change

Taking into Account Multiple Dependencies: we suggest to reformulate a directed (and potentially cyclic) graph into a Bayesian network. Indeed, the

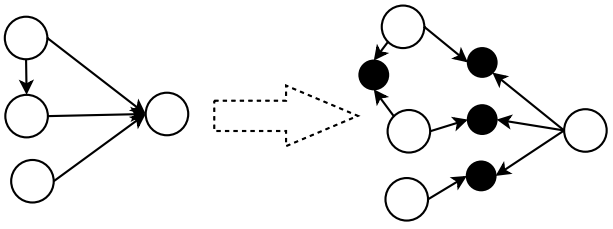


Fig. 1. Example of representation change (BN => FBN)

Bayesian network formalism is a well known and grounded theoritical and practical framework.

However, two problems arise with such a representation change: (1) the cost of taking into account the multiple dependencies which exist for a variable (i.e. computing $P(Y|X_1, X_2, ..., X_n)$, resulting in 2^n parameters when dealing with binary variables) and (2) reformulating a directed cycle within a Bayesian network.

Our solution rely on simplifying the constrains due to multiple dependencies. Indeed, multiple dependencies are decomposed by dealing with them two by two (i.e. taking separately $P(Y|X_1), P(Y|X_2), ..., P(Y|X_n)$ (resulting in $2n$ parameters for binary variables) as well as introducing constraints during the transformation process).

3.2 Taking into Account Multiple Dependencies Two by Two

Let V_1, V_2, ..., V_n, with n discrete random variables, of modality m_1, ..., m_n.

We assume that $p_i = P(V_i)$ are known (vector of size m_i), for all i, and some $p_{i,j} = P(V_j|V_i)$, $j \in I_i \subset \{1, ..., n\}$ ($p_{i,j}$ is a matrix of size (m_i, m_j)).

This model can be represented by a graph where nodes are random variables V_i and edges $a_{i,j}$ that represents the $p_{i,j}$. The conditional probabilities induce a structure that is not constrained (for instance, there may exist directed cycles). In order to simplify the notation, we introduce the notion of **Flattened Bayesian Network** (or FBN) to designate the networks that are described in the following of the paper. Figure 1 shows an example of representation change from a graph into a Flattened Bayesian Network.

Reformulating into Bayesian Network Formalism: Additional Variables and Axioms: For each pair of dependent variables (V_i, V_j), we add an additional variable which parents are V_i and V_j. This provides two advantages: (1) limiting the complexity of multiple dependencies (at the cost of approximation), (2) avoiding directed cycles (in the new formalism, all edges target additional variables). Once this reformulation is completed, inference is made possible thanks to one of the several inference algorithm of Bayesian network.

Each variable V_i from the original graph is mapped into a variable of the Bayesian network, with the same modality, noted V_i (as before).

Each edge $a_{i,j}$ is mapped into an additional boolean variable in the Bayesian network, noted $A_{i,j}$. The $A_{i,j}$ have exactly two parents in the Bayesian network, namely V_i and V_j (i.e. a V-structure). These variables are *artificially* observed in order to induce a dependency between the variables V_i and V_j (observation values are assigned to "true").

Once the additional variables are added, conditional probabilities must be computed as a last step to the transformation process, that is to compute the $P(A_{i,j}|V_i, V_j)$. Let's introduce the following notations:

- Let $K_j = \cup_i \{A_{i,j}\}$;
- Let $K = \cup_j K_j$. Let $L \subset K$. We note $L = true$ the event $\forall A \in L, A = true$.

Now, we shall define an axiomatic system to satisfy. The goal is to make the probabilities $P(A_{i,j}|V_i, V_j)$ reach a fixed point (i.e. stable). This fixed point is reached thanks to an EM-inspired iterative algorithm which is described in the following. Satisfying this axiomatic system garantees a coherent network behavior with respect to the dependencies taken two by two (compared to the behavior of a classic network).

The first axiom named "behavior axiom" determines the influence of a variable onto another. This axiom specifies a property defined from $K = true$, i.e. $\forall i, j\ A_{i,j} = true$. Then, this implies a coupled equation system. The **behavior axiom** is defined as follow:

$$\forall i, j\ P(V_j|V_i, K = true) = p_{i,j} \tag{1}$$

Secondly, the information contained in a probability distribution is linked to the difference between this distribution and the *a priori* distribution. We then introduce a second axiom named "not adding information" which states that adding additional variables do not bring information to the network. Then, this axiom implies local constrains on the $P(A_{i,j}|V_i, V_j)$, i.e. independently taking into account the $A_{i,j}$. The **not adding information axiom** is defined as follow:

$$\forall j, P(V_j|K = true) = p_j \tag{2}$$

Let's now describe the iterative process that satisfies the axioms. For more details on the equation system induced by the axioms, the reader can refer to the appendix at the end of this paper.

Satisfaction Mechanism of the Axiomatic System: For each iteration, there is an inter-dependency problem when computing the probabilities $P(A_{i,j}|V_i, V_j)$[4] Indeed, if an element of the matrix $P(A_{i,j}|V_i, V_j)$ is modified, then the axioms may be invalidated for another dependency. In pratical, we check that the system satisfy the axioms once all the matrices are calculated. We iterate the process (updating the matrix) until it converges. This is achieved thanks to an EM-inspired iterative algorithm which is concerned with the axioms and is defined as follow:

[4] This is even more true with directed cycles.

– step E: $\forall i, j\ q_{i,j} = P(V_j|V_i, K \setminus \{A_{i,j}\} = true)$;
– step M: compute $P(A_{i,j}|V_i, V_j)$ wrt. $q_{i,j}$.

At this point, this algorithm is not sufficient to make $P(A_{i,j}|V_i, V_j)$ converge. Thus, we have to limit the influence between variables through "limited update" constraints. In the following, we present the mechanisms which are necessary to the algorithm that will be described in the next section.

Convergence Parameter: Link "Strength". For each arc between two variables, we introduce a new term, namely "strength", which determines the influence of one variable upon another. A zero strength means that the variable has no direct influence (i.e. same as removing the additional variable). The strength is expressed by f, function defined on the set of additional variables $A_{i,j}$. $f(A_{i,j}) = (f_1(A_{i,j}), ..., f_{m_i}(A_{i,j}))$ is a vector of size m_i (number of modality for the variable V_i), and $f_k(A_{i,j}) = 1 - H_k(P(A_{i,j}|V_i, V_j))$ where $H_k(P(A_{i,j}|V_i, V_j))$ is the entropy of line k ($P(A_{i,j}|V_i, V_j)$ is a matrix).

Updating Criterion Used to Converge: Limiting the Direct Influence of Variables Thanks to the Strength Term. In order to compute the influence of a variable i on another variable j, we have to take into account both the direct influence (i.e. through an additional variable A_{ij}) and indirect influence (i.e. through the other variables of which i and j both depend).

For some configurations however, influences will compensate each other so that they will both tend to a limit state (probability will tend to 0 or 1), making it difficult to take them into account any further. As a matter of fact, we shall then face (1) possibly infinite convergence towards 0 or 1 and (2) computational problem related the computer accuracy (the latter being the most important in practical).

In order to solve this problem, we compute a maximum threshold for the strength which is defined for every pairs of variables and for every modality of the source variable such as:

Let $f_k^0(i, j) = f_k(A_{i,j})$ when $\forall i, j\ q_{i,j} = p_j$.

This threshold is meant to be used as the link strength if there is no indirect influence. Hence, the iterative algorithm we present in the next section must satisfy for each step: $\forall i, j\ f_k(A_{i,j}) \leq f_k^0(i, j)$ (refer to algorithm 2 in the next section).

4 Representation Change Algorithm

In this section, we present two complementary algorithms that perform the desired representation change. The first algorithm makes the system converge (i.e. N iterations until convergence) while the second algorithm makes sure that the representation change is performed with respect to the axioms for any pair of variables (i.e. a single iteration which may or may not lead to convergence).

4.1 Algorithm 1: Do *N* Iterations Until Convergence

while $P(A_{i,j}|V_i, V_j)$ haven't converged (distance from the term before is more than a given threshold) or while the number of iterations have not reached a maximum **do**
  call algorithm 2
  compute the distance between new and old probabilities
end while

4.2 Algorithm 2: Do an Iteration for All the Variables Pairs

1: **for all** pairs of variables V_i, V_j such that there exists a dependency $V_i -> V_j$ **do**
2: **if** first iteration **then**
3: Set all the additional variables as unobserved.
4: Affect the $q_{i,j} = P(V_j)$.
5: **else**
6: Set the variable $A_{i,j}$ unobserved and the other additional variables observed to true
7: Calculate the $q_{i,j} = P(V_j|V_i, K \setminus \{A_{i,j}\} = true)$ using an inference in the Bayesian network. These conditional probabilities represents the direct influence(without the link through variable $A_{i,j}$) of V_i on V_j.
8: **end if**
9: Apply the equations of the first axiom in order to determine the $P(A_{i,j}|V_i, V_j)$ with a multiply constant for each line i
10: **for all** The lines k of the matrix $P(A_{i,j}|V_i, V_j)$, calculate the "strength" $f_k = f_k(A_{i,j}) = 1 - H_k(P(A_{i,j}|V_i, V_j))$ of the link $i -> j$. **do**
11: **if** First iteration **then**
12: $f_k^0(i, j) = f_k(A_{i,j})$
13: **else**
14: **if** $f_k > f_k^0$ **then**
15: Calculate by dichotomy the $0 \leq y \leq 1$ such as $f_k(A_{i,j}^y))f_k^0$,(i.e. all the coefficients of the matrix are powered by y). This is done in order to "smooth" the parameters to increase the entropy and then decrease the "strength".
16: **end if**
17: **end if**
18: **end for**
19: Apply the equations of the second axiom to determine the multiply constants
20: Compute the matrix $P(A_{i,j}|V_i, V_j)$
21: **end for**

In the next section, we show some experiments that rely on this algorithms.

5 Experiments

5.1 Experimental Setup

In order to experimentally validate our approach, we conducted some experiments on the learnability of the networks after a representation change (i.e. flattened Bayesian networks). Our experimental setup is defined as follow:

- a generator network which can either be a flattened Bayesian network (exp. 1) or a classic Bayesian network (exp. 2). In both experiments, the number of nodes in the generator and learnable networks is fixed (in the case of flattened Bayesian network, we do not count the additional nodes built by our representation change algorithm).
- a set of learning networks that covers both *all* the possible classic Bayesian networks and flattened Bayesian networks structures with the same number of nodes than the generator (i.e. learning is exhaustive for all structures with a given size).

So as to get a good approximation of the results, we compute N data sequence from M random initializations for the generator network. As a consequence, we perform $N * M$ learning sessions for each target network ($20 \leq N * M \leq 50$).

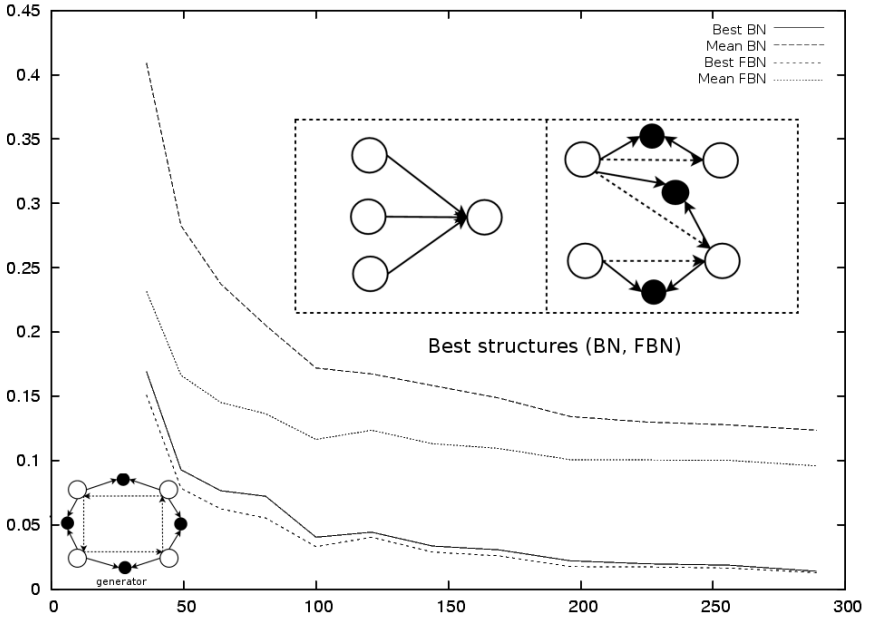


Fig. 2. Results using a Flattened Bayesian Network generator. The X-axis shows the number of examples used for learning. The Y-axis shows the Kullback-Leibler distance between the learned joint distribution and the one that was used to generate the learning data. The generator network is shown on the figure (lower-left). The best performing Bayesian and flattened Bayesian networks for 50 examples are also shown on the figure (up)

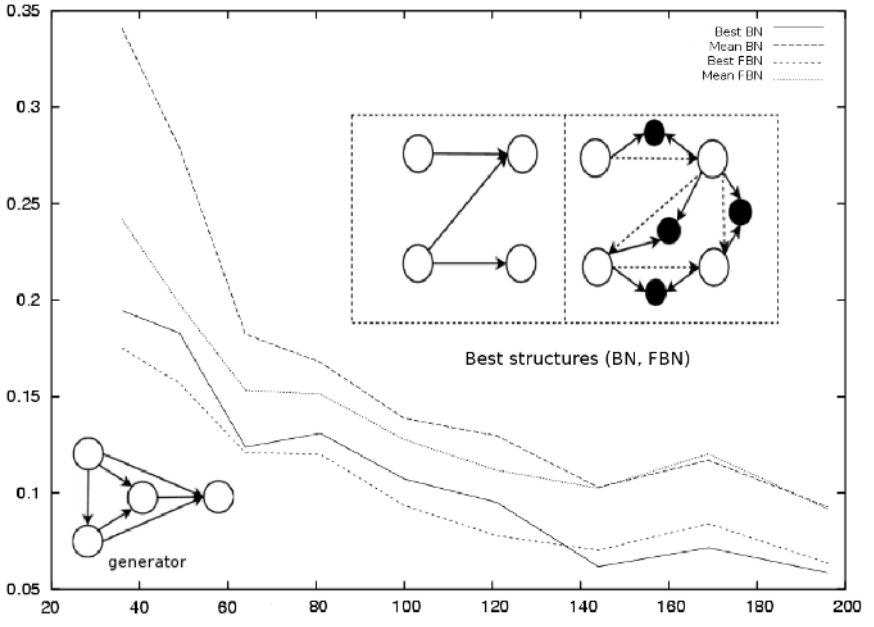


Fig. 3. Results using a Bayesian Network generator. The X-axis shows the number of examples used for learning. The Y-axis shows the Kullback-Leibler distance between the learned joint distribution and the one that was used to generate the learning data. The generator network is shown on the figure (lower-left). The best performing Bayesian and flattened Bayesian networks for 50 examples are also shown on the figure (up)

The error is defined as the Kullback-Leibler distance between the joint distribution of a given target network and the distribution of the generator network. In the scope of this paper, the network size for all experiments is limited to 4 so that it is possible to evaluate the performance for all possible structures. As a matter of fact, the number of possible structures grows more than exponentially in function of the network size, which makes computation quickly prohibitive.

5.2 Experiment 1: Learning from Data Generated by a Flattened Bayesian Network

Firstly, we study the behavior of flattened Bayesian networks in the most favorable setup, i.e. when learning on data generated by a flattened Bayesian network. In this experiment, the generator is a 4-node cyclic flattened Bayesian network. Figure 2 shows this generator as well as the results obtained with both all the flattened Bayesian networks and classic Bayesian network that contains 4 nodes.

This figure shows that the flattened Bayesian networks always perform better for average *and* best performances. However, learning performance tends to be the same as the number of examples increases ($\geq$ 250). Flattened Bayesian networks are thus relevant when learning from such data. Moreover it should be noted that the best performing flattened Bayesian network is structuraly

different from the generator, meaning that the more reliable structure when few examples are available is not the very structure of the generator.

5.3 Experiment 2: Learning from Few Examples

Secondly, we choose a 4-node classic Bayesian network as data generator (cf. fig. 3). As a consequence, learning with flattened Bayesian networks faces the worst case since the generator's joint probability can be anything. As a matter of fact, flattened Bayesian network are supposed to be better for *some* distributions (unknown at this stage of our research).

Figure 3 shows the results with respect to the experimental setup described earlier. The important result is that the flattened Bayesian networks show the best results both in average and for the best when there are few examples to learn from. However, classic Bayesian networks become better as the number of examples grow. These results show clearly that flattened Bayesian network pay for the advantage of learning speed with a loss in accuracy in the long term (i.e. compromise between a fast learning curve againt non-accurate learning in the long term).

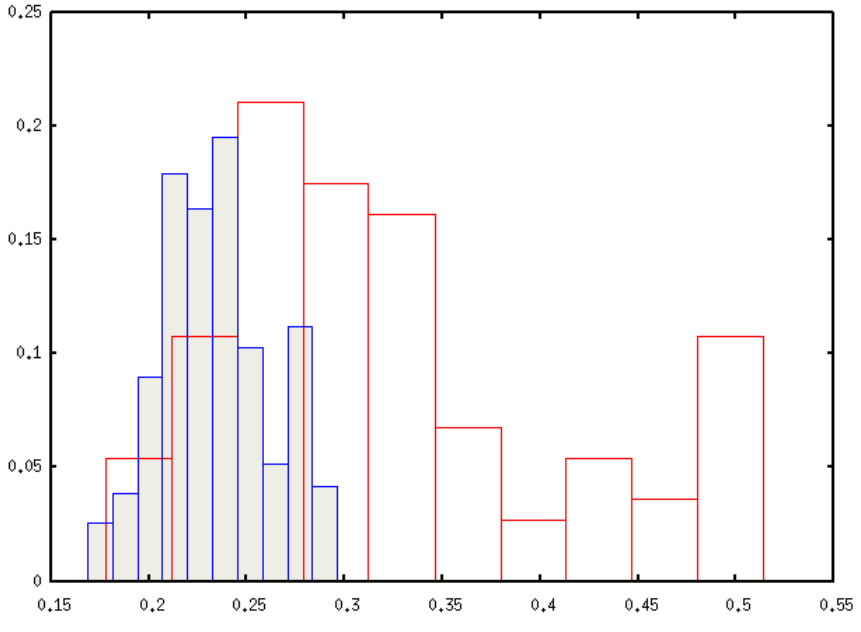


Fig. 4. Distribution (y-axis) wrt. the learning error (x-axis) for both classical (white bars) and flattened (grey bars) Bayesian networks. Learning performance for the flattened Bayesian networks are much more structuraly-independent than for classic Bayesian networks

5.4 Discussion

According to the results obtained earlier, it appears that the best networks are also the simplest ones. Thus, it seems more relevant to learn with a simple yet inadequate structure rather than with a more complex structure that is closer to the generator: this can be seen as an explanation for the good learning capabilities of flattened Bayesian networks. Figure 4 tends to confirm this assertion by showing the distribution of classic and flattened Bayesian networks according the learning performance for a given number of examples (here arbitrarily fixed to 50) in experiment 2. Indeed this figure shows that flattened Bayesian network are much less sensitive to structural variations than classic Bayesian networks.

6 Conclusions

In the scope of this paper, we were interested in the transformation of a graph (in practical, a hierarchical and factorial HMM) into a Bayesian network according to some given constraints in order to reformulate the multiple dependencies and cycles inherent to such a representation. We presented an algorithm that performs a representation change in order to build a flattened Bayesian network. We also presented the axioms that are used to provide a relevant model of reformulated multiple dependencies. This model is based on a compromise between accurateness and learning speed which is achieved by taking into account multiple dependencies by modeling variables only by pairs.

In order to study the behavior of flattened Bayesian networks, we performed two experiments that successively showed (1) the learning behavior with a cyclic flattened Bayesian network generator and (2) the learning behavior with few examples. Thanks to these experiments, we have shown that flattened Bayesian networks are especially good when learning from few examples, compared to classic Bayesian networks.

Given our original motivation, i.e. map representation in mobile robotics, the results we obtained are very promising since it has been observed that flattened Bayesian network have the following properties:

- it is possible to modelize cycles one may encountered when dealing with factorial and hierarchical HMM;
- learning is performed more quickly with fewer examples. Of course, this results from a compromise that implies a loss of accuracy in the long term. However, in the scope of mobile robotics, this compromise is worthwhile since a robot often deals with few or biased examples to build a representation of the environment.

In the scope of this paper, we presented some experiments that compare classic and flattened Bayesian networks. However, the representation formalism remains the one of the Bayesian network, eventhough the representation change algorithm adds additional variables. Thus, it is possible to build some hybrid representations that combine both flattened and classical Bayesian sub-networks

depending on the availability of data to learn from. As a consequence, this would make it possible to get the best of both worlds: learning from few examples with flattened Bayesian networks and precision learning with classic Bayesian networks when many examples are available.

Some issues remain to be explored. From the model viewpoint, the convergence mechanism describe in section 3.2 is based on experimental validation and may require some further theoretical investigations. From the robotic viewpoint, it is crucial to evaluate the learning behavior of flattened Bayesian networks using real-world data, i.e. such as those a robot could gather in its environment. Then, we should investigate the learning mechanism that may be used to find a relevant structure for a flattened Bayesian network, eventhough those netwoks have been shown to be less sensitive to an ill-chosen structure than classical Bayesian networks actually are.

References

1. D. Fox and J. Ko and K. Konolige and B. Stewart: A hierarchical Bayesian Approach to the Revisiting Problem in Mobile Robot Map Building Proc. of the International Symposium of Robotics Research (ISRR-03) (2003)
2. David Filliat and Jean-Arcady Meyer: Global localization and topological map-learning for robot navigation Proceedings of the Seventh International Conference on simulation of adaptive behavior: From Animals to Animats (SAB-2002), pages 131-140. The MIT Press (2002)
3. Zoubin Ghahramani and Michael I. Jordan: Factorial Hidden Markov Models Machine Learning, vol. 29. 1996, pages 245-273
4. Michael Montemerlo and Sebastian Thrun and Daphne Koller and Ben Wegbreit: FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem Proc. 10 th National Conference on Artificial Intelligence p593-598 (2002)
5. G. Theocharous and K. Murphy and L. Kaelbling: Representing hierarchical POMDPs as DBNs for multi-scale robot localization Proc. of the IEEE international Conference on Robotics and Automation (ICRA'04) (2004)
6. G. Theocharous and Sridhar Mahadevan: Approximate Planning with Hierarchical Partially Observable Markov Decision Process Models for Robot Navigation Proc. of the IEEE International Conference on Robotics and Automation (ICRA'02) (2002)
7. G. Theocharous and K. Rohanimanesh and S. Mahadevan: Learning Hierarchical Partially Observable Markov Decision Processes for robot navigation Proceedings of the IEEE Conference on Robotics and Automation (ICRA-2001). IEEE Press (2001)

Appendix: Property of the Model and Conditional Probabilities

This section details the equations obtains from the axioms.

We add to the notations above, for i, j fixed: $K' = K \setminus \{A_{i,j}\}$.

First Axiom

The first axiom, named "behavior" determines the influence of a variable on another. This axiom specify a property defined from $K = true$, that is to say from $\forall i, j\ A_{i,j} = true$. It can be expressed as: Behavior axiom:

$$\forall i, j\ P(V_j|V_i, K = true) = p_{i,j}$$

(axiom 1)

From Bayes formulae,

$$\begin{aligned} & P(A_{i,j} = true|V_i = k, V_j = l, K' = true) \\ = & \frac{P(V_j = l|V_i = k, K = true)P(A_{i,j} = true|V_i = k, K' = true)}{P(V_j = l|V_i = k, K' = true)} \end{aligned} \tag{3}$$

Finally:

$$\forall k, l, P(A_{i,j}|V_i = k, V_j = l) = \gamma_k \frac{p_{i,j}(k, l)}{P(V_j = l|V_i = k, K' = true)}$$

The proportionality coefficients γ_k, which do not have influence on the satisfaction of this property allow us to obtain the following property.

Second Axiom

The information contained in a probability distribution is linked to the difference between this distribution and the *a priori* distribution. We then introduce a second axiom named "not adding information" which states that adding additional variables will not bring information in the network. Then, this axiom implies local constraints on the $P(A_{i,j}|V_i, V_j)$, that is to say taking into account the $A_{i,j}$ independently. More precisely:

not adding information axiom:

$$\forall i, \forall k, P(V_i = k|K = true) = p_i(k)$$

Again from the Bayes formulae:

$$\begin{aligned} & P(V_i = k|K = true) \\ = & \frac{P(A_{i,j} = true|V_i = k, K' = true)P(V_i = k|K' = true)}{P(A_{i,j} = true|K' = true)} \\ = & P(V_i = k|K' = true) \\ & \frac{\sum_l P(A_{i,j} = true|V_j = l, V_i = k)p(V_j = l|V_i = k, K' = true)}{P(A_{i,j} = true|K' = true)} \end{aligned} \tag{4}$$

Futhermore, we have from above:

$$P(A_{i,j} = true|V_j = l, V_i = k) = \gamma_k \frac{p_{i,j}(k, l)}{P(V_j = l|V_i = k, K' = true)}$$

Hence:

$$
\begin{aligned}
& P(V_i = k | K = true) \\
= & \frac{P(V_i = k | K' = true) \sum_l \gamma_k \frac{p_{i,j}(k,l)}{P(V_j = l | V_i = k, K' = true)} p(V_j = l | V_i = k, K' = true)}{P(A_{i,j} = true | K' = true)} \\
= & \frac{P(V_i = k | K' = true) \sum_l \gamma_k p_{i,j}(k,l)}{P(A_{i,j} = true | K' = true)} \\
= & \frac{P(V_i = k | K' = true) \gamma_k}{P(A_{i,j} = true | K' = true)}
\end{aligned}
$$

Finally:

$$\forall k, \gamma_k = P(A_{i,j} = true | K' = true) \tag{5}$$

All the γ_k are equals. This constant does not have influence on the wanted properties and is then chosen in order to have all the probabilities between 0 and 1, and secondly for numerical considerations. More precisly it is chosen such as

$$max_{j,l} P(A_{i,j} = true | V_i = k, V_j = l) = 1$$ ■

Hierarchical Heuristic Search Revisited

Robert C. Holte, Jeffery Grajkowski, and Brian Tanner

University of Alberta, Computing Science Department,
Edmonton, Alberta, Canada
{holte, grajkows, btanner}@cs.ualberta.ca

Abstract. Pattern databases enable difficult search problems to be solved very quickly, but are large and time-consuming to build. They are therefore best suited to situations where many problem instances are to be solved, and less than ideal when only a few instances are to be solved. This paper examines a technique - hierarchical heuristic search - especially designed for the latter situation. The key idea is to compute, on demand, only those pattern database entries needed to solve a given problem instance. Our experiments show that Hierarchical IDA* can solve individual problems very quickly, up to two orders of magnitude faster than the time required to build an entire high-performance pattern database.

1 Introduction

Pattern databases were introduced [3, 4] as a method for defining a heuristic function to be used by heuristic search algorithms such as A* [9] and IDA* [14]. They have proved very valuable. For example, they are the key breakthrough enabling Rubik's Cube to be solved optimally [15], they have advanced the state of the art of solving the sequential ordering problem [11], and have enabled the length of solutions constructed using a macro-table to be very significantly reduced [10]. They have also proven useful in heuristic-guided planning [6].

A pattern database is defined by a goal state and an abstraction, ϕ, that maps the given state space, S, to an abstract state space $\phi(S)$. The states in $\phi(S)$ are called abstract states or patterns. A pattern database is a lookup table with an entry for each pattern – the entry for pattern P is the distance in $\phi(S)$ from P to the goal pattern, $\phi(goal)$. Given a pattern database, the heuristic value, $h(s)$, for a state $s \in S$ is computed by looking up the entry for $\phi(s)$ in the pattern database. Because ϕ is an abstraction, $h(s)$ is guaranteed to be an admissible, monotone heuristic [12].

A pattern database is built by finding a shortest path to the goal pattern for every pattern in $\phi(S)$. Typically this is done by running a breadth-first search backwards from the goal pattern until $\phi(S)$ is fully enumerated.

Building an entire pattern database as a preprocessing step has two disadvantages. The first is the time it takes to build the pattern database. For example, the "7-8" additive pattern database for the 15-puzzle in [7] takes approximately 3 hours to build and the high-performance pattern database for (17,4)-TopSpin

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 121–133, 2005.

in [8] takes approximately 1 hour to build. The second disadvantage is the size of the pattern database. In solving a single problem, only a tiny fraction of the pattern database is needed, and therefore most of the memory allocated for the pattern database, and the time needed to build it, is wasted.

Both disadvantages disappear, to some extent, if the same pattern database is used to solve many problem instances. For example, if 3 million 15-puzzle instances are solved using the "7-8" pattern database, the majority of its entries would be needed and the time to build the pattern database would amount to less than 10% of the total solution time.

On the other hand, there are circumstances in which the cost of building an entire pattern database cannot reasonably be amortized over a large number of problem instances. The obvious such circumstance is when only one or a few problem instances need to be solved, such as when building a macro-table [10], or when solving multiple sequence alignment problems [17]. In this case the time to build the pattern database will dominate the time to solve the problems.

Another circumstance in which the cost of building a pattern database cannot be amortized is when there are many instances to solve but it is impossible to use the same pattern database to solve them because they have different goals (and no simple transformation is possible), or because the operators or their costs have changed. As will be shown below it can also happen that, even though it is possible to use the same pattern database for all the problem instances, it is advantageous, time-wise, to use a different, custom-selected pattern database to solve different instances.

In this paper we examine a technique - hierarchical heuristic search - that aims to minimize the time and space overhead of using a pattern database by computing only those entries of the pattern database that are actually needed to solve a given problem instance. The idea of on-demand calculation of pattern database entries by hierarchical heuristic search was introduced in [12]. The abstraction technique there was so costly, in terms of both time and space, that it needed to be amortized over a large number of problem instances and therefore offered little or no advantage over pattern databases. The starting point for the present paper is the observation that the abstraction technique used for pattern databases requires negligible space and time, and therefore raises the possibility of realizing the great potential advantages of hierarchical heuristic search over pattern databases for solving individual problem instances. In addition to using a different abstraction technique, the present work also uses IDA* as its basic search procedure, whereas [12] used A*.

This paper reports several experiments with Hierarchical IDA* (HIDA*). The first shows that even if one abstraction, somewhat arbitrarily chosen, is used to solve all problem instances for a given state space, an average instance can be individually solved from scratch by HIDA* in minutes, compared to the one or more hours it takes to build a high-performance pattern database. Subsequent experiments show that in some state spaces a substantial additional speedup can be obtained by using multiple or customized abstractions.

2 The Hierarchical IDA* Algorithm

Pseudocode for Hierarchical IDA* (HIDA*) is given in Figure 1. The lines highlighted in bold font indicate the differences from normal IDA*. To improve performance our actual implementation is somewhat more complex, but this figure captures the central ideas.

The defining characteristic of hierarchical heuristic search is seen in the $h(s, goal)$ function in Figure 1. To compute a heuristic for state s, a recursive call to the search algorithm is made to compute the exact distance between the abstraction of s, $\phi(s)$, and the abstraction of the goal state, $\phi(goal)$. Search at an abstract level is guided by distances computed at an even more abstract level (in the figure the symbol ϕ is used to indicate the function that moves from the current level to the next more abstract level; an alternative notation would have had a different symbol, ϕ_i, for each level).

```
HIDA*(start, goal)
  bound ← h(start, goal)
  Repeat until goal is found:
     bound ← DFS(start, goal, 0, bound)
  For all states s on the solution path:
     cache[s] ← distance from s to goal
     mark cache[s] as an exact distance
-------------------------------------------------
DFS(s, goal, g, bound)
  If s == goal: exit with success
  g ← g + 1
  newbound ← ∞
  Iterate over x ∈ successors(s):
  // P-g caching
     cache[x] ← max(cache[x],bound-g,h(x,goal))
     f ← g + cache[x]
  // Optimal path caching
     If (f == bound) and (cache[x] is an exact distance):
        exit with success
     If f ≤ bound: f ← DFS(x, goal, g, bound)
     If f < newbound : newbound ← f
  Return newbound
-------------------------------------------------
h(s, goal)
  If at the top abstraction level, return 0
  If cache[φ(s)] is not an exact distance:
     HIDA*(φ(s),φ(goal))
  Return cache[φ(s)]
```

Fig. 1. Pseudocode for Hierarchical IDA*

Whenever the exact distance from a state to the goal is determined at any level other than the base level (the original search space), it is stored in a cache so that it need not be recomputed if it is required again. This is done by the bold lines of pseudocode in the HIDA* function in Figure 1.

Exact distances to the goal stored in $cache[x]$ for abstract state, x, are actually used for two different purposes. First, $cache[x]$ is used as the value for $h(s, goal)$ for any less abstract state, s, for which $\phi(s) = x$. Second, $cache[x]$ is used in the optimal path caching method that was introduced, for A*, in [12]. If x is reached by a path of length g during a search within x's abstract level and $cache[x]$ is an exact distance to goal, it is treated as if the goal had been reached by a path of length $g + cache[x]$. See the bold lines after the "Optimal path caching" comment in the DFS function in Figure 1.

In addition to storing exact distances to the goal, the cache is also used to store estimated distances to the goal generated by the "P-g" caching technique that was introduced, for A*, in [12]. "P-g" caching improves (increases) the heuristic values for abstract states that are expanded during a search, thereby improving the efficiency of subsequent searches that happen to reach these states. One possible implementation of "P-g" caching for HIDA* is shown by the bold lines after the "P-g caching" comment in the DFS function in Figure 1.

In all experiments the memory used for the cache was limited to 1 Gigabyte. The implementation of the cache in hierarchical search is less efficient than the hash table used to implement pattern databases because it is not known ahead of time which entries, or even how many entries, will be put into the hierarchical search cache[1]. By contrast, the exact set of patterns that will index a pattern database is known ahead of time. This is enormously beneficial in terms of space because a perfect hash function (collision free, no gaps) can be used, meaning that nothing identifying the pattern needs to be stored as part of an entry. Pattern database entries therefore only contain distances, typically needing only one byte per entry. It is not possible to develop a perfect hash function for the hierarchical search cache, so the cache must store a unique identifier for each pattern along with its distance, which increases the size of an entry very substantially (e.g. from one byte to eight for our 15-puzzle implementation). Not having a perfect hash function also slows down access, since collisions can occur and must be detected and resolved.

On a modern PC (AMD Athlon, 2.1GHz) our code generates approximately 4.5 million nodes per second at the base level and 1.5 million nodes per second at the abstract levels. The difference in speed is because the cache operations are done at each abstract level but not at the base level.

[1] This is true only of the lower levels in the abstraction hierarchy. For the upper levels it is virtually certain that almost all possible entries will be generated. For example, in the 15-puzzle experiment in the next section over 90% of the possible entries were generated at each of levels 4-8 in Table 1.

3 State Spaces Used in the Experiments

Four state spaces are used in these experiments: the 15-puzzle, a novel variant called the Macro-15 puzzle, (17,4)-Topspin, and the 14-Pancake puzzle. For each state space 100 randomly generated solvable instances were used for testing.

The 15-puzzle is included in our experiments because it is a standard benchmark for heuristic search methods. It is not actually a good example of the circumstances in which to use HIDA* because a very good, efficiently computed heuristic (Manhattan Distance) is known for it, and with modern search methods individual problems can be solved from scratch very quickly [1].

The Macro-15 puzzle is a novel variation on the 15-puzzle inspired by the fact that in the physical puzzle it takes the same effort to slide any partial row or partial column of tiles one position towards the blank as it takes to slide a single tile. Thus, in the 4x4 Macro puzzle used here there are 6 possible moves in every state (because there are 3 tiles in the same row as the blank and 3 tiles in the same column as the blank, and any tile in the same row or column as the blank can be the endpoint of the group that is moved). We call this state space the Macro-15 puzzle because its additional moves are "macro" moves in the 15-puzzle. Solution lengths in the Macro-15 state space for the 100 standard test problems used for the 15-puzzle [14] range from 27 to 38 with the median and average solution length being 32. By contrast in the normal 15-puzzle these problems' solution lengths range from 41 to 66, with a median and average length of 53. Note that Manhattan Distance is not an admissible heuristic in this space, and additive pattern databases [7] cannot be used for it.

The (N,K)-TopSpin puzzle has N tokens arranged in a ring. The tokens can be shifted cyclically clockwise or counterclockwise. The ring of tokens intersects a region K tokens in length which can be rotated to reverse the order of the tokens currently in the region. In our encoding we ignore the cyclic shifts and only count reversals. Therefore the only moves are to reverse any K adjacent tokens, where adjacency is defined cyclically. We used $N = 17$ and $K = 4$, but the effective number of tokens is only 16 because one of the tokens is used as a fixed reference point and therefore is effectively stationary. [2] shows that all 16! states are reachable. In order to reduce the number of transpositions, if two moves act on non-intersecting sets of positions we force them to be done in a particular order. This reduces the branching factor to 8.

In the N-Pancake puzzle [5] a state is a permutation of N tokens $(0, 1, ...N - 1)$. A state has $N - 1$ successors, with the k^{th} successor formed by reversing the order of the first $k + 1$ positions of the permutation $(1 \leq k < N)$. We used N=14, which has 14! states. Although this space is smaller than the others its much larger branching factor makes it roughly the same difficulty to search.

4 Using One Abstraction Hierarchy for All Problems

Hierarchical search requires an abstraction hierarchy – a sequence of abstractions defining the mappings from one level of abstraction to the next. In our state

Table 1. "Default" Abstraction Hierarchy for the 15-puzzle and the Macro-15 puzzle

8	•	•	•	•	•	•	•	•	15
7	•	•	•	•	•	•	•	14	15
6	•	•	•	•	•	•	13	14	15
5	•	•	•	•	•	12	13	14	15
4	•	•	•	•	11	12	13	14	15
3	•	•	•	10	11	12	13	14	15
2	•	•	9	10	11	12	13	14	15
1	•	8	9	10	11	12	13	14	15
base	1-7	8	9	10	11	12	13	14	15

spaces an abstraction is defined by mapping some of the tiles/tokens to a "don't care" symbol. If the tokens mapped to "don't care" by ϕ_1 are a superset of the tokens mapped to "don't care" by ϕ_2 then the space defined by ϕ_1 is an abstraction of the space defined by ϕ_2. Therefore a sequence of successively more abstract spaces can be defined by partitioning the tokens into groups, $G_1, G_2, ..., G_n$ and defining ϕ_i as the abstraction in which all tokens in groups $G_1, G_2, ..., G_i$ are mapped to "don't care".

In the experiment in this section the same abstraction hierarchy is used for all problem instances of each state space. These "default" abstraction hierarchies were not carefully chosen, they were among our initial thoughts for each state space. The abstraction hierarchy used for the 15-puzzle and the Macro-15 puzzle is shown in Table 1, which is read from bottom to top, because the higher rows represent the higher levels of abstraction. The bottom row ("base" level) indicates which tile(s) each column is referring to. Each other row indicates how the tiles are mapped at a certain level of abstraction, the level being indicated by the number in the first column. For example the row with 1 in the first column shows that the first level of abstraction is defined by mapping tiles 1-7 to "don't care" (indicated by • in the table). The patterns at this abstract level are the possible ways of placing the blank and the 8 remaining tiles (tiles 8-15) in the 16 positions of the 15-puzzle. At the most abstract level (level 8) all the tiles except 15 are mapped to "don't care". The patterns at this abstract level are the possible ways of placing the blank and tile 15 in the 16 positions of the 15-puzzle.

The abstraction hierarchy used for the 14-Pancake puzzle is identical except that it has only 14 tokens and therefore only seven abstract levels. Note that "token 1" has the most volatile home position – the token in that position is changed by every operation. This abstraction therefore abstracts the tokens in volatile positions and retains the identity of tokens that can be placed in their home positions and then left unmoved by a judicious choice of operators.

The abstraction hierarchy for (17,4)-TopSpin starts by abstracting tokens 1-9 to define the first abstraction level and abstracts one token per level thereafter in increasing order (10, then 11, then 12 etc.). Token 0 is the token that is used as a reference and never moves - it is never abstracted.

Table 2. CPU times (in seconds) using the default abstraction hierarchy. "all" means clear every cache between every problem instance. "1-3" means clear levels 1-3 between every instance, never clear the higher levels. "none" means never clear any cache - this is possible only if all entries at all levels fit within the 1 Gigabyte memory limit.

State Space	Clear	Avg	Max	Median
15-puzzle	all	642	20,227	93
	1-3	596	17,827	71
Macro-15	all	132	910	84
	1-3	101	959	58
(17,4)-TopSpin	all	766	3,068	680
	none	162	1,875	89
14-Pancake	all	88	405	54
	none	31	326	4

Table 3. Comparison of the average number of cache entries (in thousands) stored by HIDA* to the number of entries (in thousands) in the full pattern database (PDB size) for the first level of abstraction. The last column expresses the average number of Level 1 cache entries as a percentage of the pattern database size.

State Space	Total	Level 1	PDB size	%
15-puzzle	10,931	2,657	4,151,347	0.06
Macro-15	7,402	787	4,151,347	0.02
(17,4)-TopSpin	8,143	3,423	57,657	5.9
14-Pancake	1,208	229	17,297	1.3

Table 2 shows the average, maximum, and median CPU times over the 100 test problems for each state space. "All" in the "Clear" column indicates that all caches are cleared completely between each problem instance. This simulates solving a single problem instance in isolation with no preprocessing or prior problem-solving experience with the abstraction. The "median" column in the "all" rows shows that the majority of individual problems can be solved in a few minutes, compared to the hour or more it takes to build high-performance pattern databases. Across the entire experiment only three problem instances, all for the 15-puzzle, take more than an hour to solve.

Table 3 shows the number of cache entries created by HIDA*, on average. "Total" is the total number of cache entries at all levels. "Level 1" is the number of cache entries for the first level of abstraction. The rightmost column shows that this is a small fraction of what would be stored in the pattern database for this abstraction – well under one-tenth of one percent for the 15-puzzle and Macro-15 puzzle.

If a small batch of problem instances with the same goal is to be solved using the same abstraction, HIDA*'s caches need not be cleared between instances: the

cache entries created for one instance will be correct for others. The abstractions being used for (17,4)-TopSpin and 14-Pancake are sufficiently coarse-grained that 1 Gigabyte allows all caches at all levels to be made large enough to hold all possible entries. Therefore, it is never necessary to clear any cache. The "none" rows in Table 2 show the CPU times for these puzzles if the 100 test problems are solved as a batch in the random order in which they were generated with no clearing of the caches. Batch-solving substantially reduces the average, median, and the maximum solution times. A batch of 18 average problems can be solved in the time it takes to build a high-performance pattern database for (17,4)-TopSpin.

For the 15-puzzle and Macro-15 puzzle abstractions levels 1, 2, and 3 are sufficiently fine-grained that they must be cleared at some point in order to solve the 100 test problems. The "1-3" rows in Table 2 show the CPU times that result if only the caches at levels 1, 2, and 3 are cleared between each problem instance. This produces a modest reduction in the time to solve problems. Batches of approximately 17 15-puzzle problems and batches of 97 Macro-15 problems can be solved in the time it takes to build a high-performance pattern database for these puzzles.

The fact that HIDA* solves tens of problem instances, on average, in the time required to build a high-performance pattern database does not rule out the possibility that HIDA* would be outperformed by a smaller pattern database when only one or a few problem instances are to be solved. To see why this will not happen, in general, consider the Macro-15 puzzle. To build the complete pattern database for the first level abstraction in Table 2 takes 2.73 hours. The pattern database based on the second level abstraction is much smaller and takes only 452 seconds to build. This is still substantially more than the time it takes HIDA* to solve a single Macro-15 problem on average. A pattern database based on an even coarser abstraction would take fewer than 100 seconds to build but would provide such poor heuristic guidance for the base level search that the time to solve a problem would far exceed HIDA*'s. As a general rule, a pattern database that can be fully computed in a time less than HIDA*'s will offer much weaker guidance than HIDA*'s first level of abstraction and therefore have higher problem-solving runtimes.

5 Multiple Abstractions

[13] shows that for a fixed amount of memory, taking the maximum of several smaller pattern databases outperforms using a single large pattern database. This technique can be applied to hierarchical heuristic search by using multiple abstractions instead of just one at one or more of the abstraction levels. However, it is not obvious if this will lead to improved performance for hierarchical heuristic search because, unlike in the pattern database studies, the time required to calculate the entries for the additional abstractions is now counted as part of the execution time.

Table 4. CPU times (in seconds) using multiple abstractions

State Space	Clear	Avg	Max	Median
15-puzzle	all	131	1,047	78
	none	31	364	7
Macro-15	all	88	392	65
	none	17	162	5
(17,4)-TopSpin	all	982	2,394	919
	none	88	927	50
14-Pancake	all	95	329	72
	none	10	197	2

In this section we define multiple abstractions only at the first level of abstraction. Each of those abstractions then has only one abstraction above it, created by abstracting one additional tile/token, and those abstractions each have only one above them, etc. As in [13], in computing $h(s)$ the calculation of the maximum value given by the different abstractions is aborted if a value is returned that is large enough to ensure that $f(s) = h(s) + g(s)$ exceeds IDA*'s current depth bound.

For (17,4)-TopSpin and the 14-Pancake puzzle, two first-level abstractions are used: the default abstraction from the previous section and a complementary one. For (17,4)-TopSpin the complementary abstraction abstracts tokens 8-16 at the first level (the default abstraction abstracts tokens 1-9) and then abstracts one additional token per level in decreasing order (7, then 6, then 5 etc.). Similarly, the complementary abstraction for the 14-Pancake puzzle abstracts tokens 7-13 (the default abstracts tokens 0-6) and then abstracts one additional token per level in decreasing order. The results are shown in Table 4. The "all" and "none" rows in Table 4 have the same meaning and can be directly compared to the corresponding rows in Table 2. The multiple abstractions in this experiment increase the CPU time for solving individual problems in isolation (the "all" rows) but significantly decrease the time for solving small batches of problems (the "none" rows).

For the 15-puzzle and the Macro-15 puzzle, the default abstractions fill available memory, so there is no room available for additional abstractions. Instead, we use four first-level abstractions that are each considerably smaller than the default. One of them abstracts 8 tiles, the others abstract 9 tiles (the default abstracts only 7 tiles). Comparing the "all" rows in Table 4 to the corresponding rows in Table 2 we see that individual problems are solved much more quickly using multiple abstractions. The average time for solving individual problems, 131 seconds for the 15-puzzle and 88 seconds for Macro-15, is two orders of magnitude less than the time required to build a high-performance pattern database for these puzzles. The multiple abstractions used here are sufficiently coarse-grained that 1 Gigabyte is enough to create perfect hash tables for all caches at

all levels. This enables batches of problems to be solved without ever clearing any caches, producing the results shown in the "none" rows in Table 4. In both spaces the entire batch of 100 test problems is solved in well under an hour.

6 Customized Abstractions

The first experiment showed that individual problem instances can be solved quickly using a default abstraction hierarchy. This section shows that this performance can be significantly improved, in some state spaces, by tailoring the abstraction hierarchy to each instance. In this experiment all caches are cleared between each instance.

For the 15-puzzle and Macro-15 puzzle we use a simple method for creating the customized abstraction hierarchy. The key idea is to choose a good order in which the tiles will be abstracted. The first level abstracts the first seven tiles according to the order, and each level after that abstracts the next tile in the ordering. The tile ordering we used is based on each tile's Manhattan distance, i.e. the number of moves required to get the tile from its position in the start state to its goal position. The tiles are sorted in increasing order of this distance, with ties broken arbitrarily.

78 of the 15-puzzle problems are solved more quickly with the customized abstraction than with the default abstraction. The 22 that are slower are all "easy" problem instances. The hardest problem instances have all been sped up substantially by using a customized abstraction; some now run almost 50 times faster than before. The longest-running instance now takes 1,517 seconds. The average time to solve a problem instance is reduced from 642 to 99 seconds, and the median time drops from 93 to 42 seconds. These results are significantly better than the results with generic multiple abstractions (row "all" in Table 4).

The Macro-15 puzzle also benefits significantly from custom abstractions, although not as much as the 15-puzzle. Average solution time is reduced to 99 seconds from 132, and the median drops to 64 seconds from 84.

For (17,4)-TopSpin and the 14-Pancake puzzle numerous methods of custom abstraction were explored. For the 14-Pancake puzzle none outperformed the default abstraction. For (17,4)-TopSpin we identified an abstraction[2] that was significantly better than the default for certain problems. However, there was no obvious rule to decide which abstraction to use on a given problem instance. Our solution was to compute $h(start)$ using each of the abstractions and then use the abstraction that gave the higher value to solve the instance. This is a rather expensive selection rule, because the cache entries created when computing $h(start)$ using the first abstraction have to be cleared in order to compute $h(start)$ using the second abstraction, and then have to be recomputed if the first abstraction is chosen for solving the problem. This overhead must be in-

[2] The first level abstracts tokens 8-16, subsequent levels eliminate one additional token in decreasing order (7 then 6 then 5 etc).

cluded in the total time to solve a problem, and doing so leaves the average and median solution times virtually the same as always using the default abstraction. However, this method does reduce the time to solve the most difficult problem from 3068 to 2304 seconds.

7 Related Work

[17] observes that the pattern database entry for $\phi(s)$ is not needed if

$$d(\phi(s), \phi(goal)) + d'(\phi(start), \phi(s)) > U$$

where $d(x, y)$ is the true distance from x to y, $U \geq d(start, goal)$, and $d'(x, y) \leq d(x, y)$. Given an upper bound, U, on the solution cost in the original space and a function, $d'(x, y)$, that never overestimates distance in the abstract space, [17] runs $A*$ backwards from the abstract goal state until it has enumerated all abstract states $\phi(s)$ with $d(\phi(s), \phi(goal)) + d'(\phi(start), \phi(s)) \leq U$. The resulting table of abstract distances is called a space-efficient pattern database (SEPDB).

If the abstraction used for the SEPDB is used to define HIDA*'s first abstract level and search at this level is guided by the same heuristic in both systems, HIDA*'s first-level cache will always contain a subset of the SEPDB entries. The SEPDB hash table, like HIDA*'s caches, must store pattern identification information along with the distance information when there is not sufficient memory to store all possible entries for the full pattern database. Thus, SEPDB's memory needs cannot be less than HIDA*'s.

To see precisely how SEPDB's memory requirements compare to HIDA*'s we ran SEPDB using the default abstractions for our state spaces. The second abstract level was used as the heuristic to guide SEPDB's A*. The resulting SEPDB is therefore the counterpart of HIDA*'s first level cache. To make the comparison as favourable to SEPDB as possible the upper bound it was given was the actual solution length for each problem instance. The results are shown in Table 5. SEPDB has at least 32% more entries than HIDA*'s first level cache, even when given a perfect upper bound. If this upper bound is increased to be just one larger than the optimal value, the size ratios for Macro-15, (17,4)-Topspin and 14-Pancake increase to 3.52, 1.75, and 4.05 respectively. For the 15-puzzle the next larger meaningful upper bound is two larger than the optimal value. In this case, the average size of the SEPDB rises to 13,515,134, which is 5.08 times larger than HIDA*'s first-level cache.

The CPU times for SEPDB and HIDA* cannot be compared in this experiment because the second level of abstraction is computed by HIDA* but assumed to be given, without computation, by SEPDB. To make a fair time comparison, a hierarchical version of SEPDB would be needed. Hierarchical SEPDB might possibly run faster than HIDA*, but, as this experiment has shown, it would require more memory.

"Reverse Resumable A*" [16], like SEPDB, computes pattern database entries by backwards A* search at the abstract level. Unlike SEPDB, it stops when

Table 5. Comparison of the sizes of HIDA*'s first-level cache and the corresponding SEPDB

Space	HIDA*	SEPDB	Ratio
15-puzzle	2,657,511	6,430,269	2.42
Macro-15	787,664	1,309,100	1.66
(17,4)-TopSpin	3,423,746	4,534,162	1.32
14-Pancake	339,328	467,237	1.38

it closes the abstract start state. If an entry is needed during the base level search that has not been generated, A* search at the abstract level is resumed until the entry is generated. This produces a subset of the SEPDB, and avoids the need for an upper bound on solution length, but requires additional memory for preserving A*'s Open list so that A* can be resumed.

8 Conclusion

This paper has shown that hierarchical heuristic search can solve individual problems very quickly, up to two orders of magnitude faster than building a high-performance pattern database. Hierarchical heuristic search is therefore preferable to pattern databases when only one or a few problem instances with the same goal are to be solved. On the other hand, pattern databases are preferable when many problem instances with the same goal are to be solved. In cases where it is unclear which method to use, the two can be used in parallel. While the pattern database is being built, hierarchical heuristic search can be applied to the problem instances, perhaps with a time limit for each problem instance. Sometimes all the instances will be solved before the pattern database is complete. If this does not happen, the remaining problems can be solved quickly using the pattern database.

Acknowledgments

This research was supported in part by a Discovery grant and a postgraduate scholarship from the Natural Sciences and Engineering Research Council of Canada and an iCORE postgraduate fellowship. We thank Rich Korf for supplying the base IDA* implementation from which our code was developed.

References

1. Andreas Auer and Hermann Kaindl. A case study of revisiting best-first vs. depth-first search. *Proceedings of the Sixteenth European Conference on Artificial Intelligence (ECAI-04)*, pages 141–145, 2004.

2. Ting Chen and Steve Skiena. Sorting with fixed-length reversals. *Discrete Applied Mathematics*, 71(1-3):269–295, 1996. Special volume on computational molecular biology.
3. J. C. Culberson and J. Schaeffer. Efficiently searching the 15-puzzle. Technical report, Department of Computer Science, University of Alberta, 1994.
4. J. C. Culberson and J. Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.
5. Harry Dweighter. Problem e2569. *American Mathematical Monthly*, 82:1010, 1975.
6. S. Edelkamp. Planning with pattern databases. *Proceedings of the 6th European Conference on Planning (ECP-01)*, pages 13–24, 2001.
7. A. Felner, R. E. Korf, and S. Hanan. Additive pattern database heuristics. *Journal of Artificial Intelligence*, 21:1–39, 2004.
8. Ariel Felner, Uzi Zahavi, Jonathan Schaeffer, and Robert Holte. Dual lookups in pattern databases. *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, 2005.
9. P.E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4:100–107, 1968.
10. I. T. Hernádvölgyi. Searching for macro operators with automatically generated heuristics. *Advances in Artificial Intelligence - Proceedings of the Fourteenth Biennial Conference of the Canadian Society for Computational Studies of Intelligence (LNAI 2056)*, pages 194–203, 2001.
11. I. T. Hernádvölgyi. Solving the sequential ordering problem with automatically generated lower bounds. *Proceedings of Operations Research 2003 (Heidelberg, Germany)*, pages 355–362, 2003.
12. R. C. Holte, M. B. Perez, R. M. Zimmer, and A. J. MacDonald. Hierarchical A*: Searching abstraction hierarchies efficiently. *Proc. Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 530–535, 1996.
13. Robert C. Holte, Jack Newton, Ariel Felner, and Ram Meshulam. Mutiple pattern databases. *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS-04)*, pages 122–131, 2004.
14. R. E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.
15. R. E. Korf. Finding optimal solutions to Rubik's Cube using pattern databases. *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 700–705, 1997.
16. David Silver. Cooperative pathfinding. *Proceedings of the First Annual Conference on Artificial Intelligence and Interactive Entertainment (AIIDE-05)*, pp. 117–122, 2005.
17. R. Zhou and E. A. Hansen. Space-efficient memory-based heuristics. *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-04)*, pages 677–682, 2004.

Multinomial Event Model Based Abstraction for Sequence and Text Classification

Dae-Ki Kang, Jun Zhang, Adrian Silvescu, and Vasant Honavar

Artificial Intelligence Research Laboratory,
Department of Computer Science,
Iowa State University,
Ames, IA 50011 USA
{dkkang, jzhang, silvescu, honavar}@cs.iastate.edu

Abstract. In many machine learning applications that deal with sequences, there is a need for learning algorithms that can effectively utilize the hierarchical grouping of words. We introduce Word Taxonomy guided Naive Bayes Learner for the Multinomial Event Model (WTNBL-MN) that exploits word taxonomy to generate compact classifiers, and Word Taxonomy Learner (WTL) for automated construction of word taxonomy from sequence data. WTNBL-MN is a generalization of the Naive Bayes learner for the Multinomial Event Model for learning classifiers from data using word taxonomy. WTL uses hierarchical agglomerative clustering to cluster words based on the distribution of class labels that co-occur with the words. Our experimental results on protein localization sequences and Reuters text show that the proposed algorithms can generate Naive Bayes classifiers that are more compact and often more accurate than those produced by standard Naive Bayes learner for the Multinomial Model.

1 Introduction

In machine learning, one of the important goals is to induce comprehensible, yet accurate and robust classifiers [1]. In classical inductive learning for text classification, each document is represented as a bag of words. That is, one instance is an ordered tuple of word frequencies or binary values to denote the presence of words. However, these words can be grouped together to reflect assumed or actual similarities among the words in the domain or in the context of a specific application. Such a hierarchical grouping of words yields word taxonomy (WT). Figure 1 is an example of word taxonomy of "Science" made by human.

Taxonomies are very common and useful in many applications. For example, Gene Ontology Consortium has developed hierarchical taxonomies for describing various aspects of macromolecular sequences, structures, and functions [2]. For intrusion detection, Undercoffer et al.[3] established a hierarchical taxonomy of features observable by the target of an attack. Various ontologies have been developed in several fields as part of Semantic Web related efforts [4].

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 134–148, 2005.

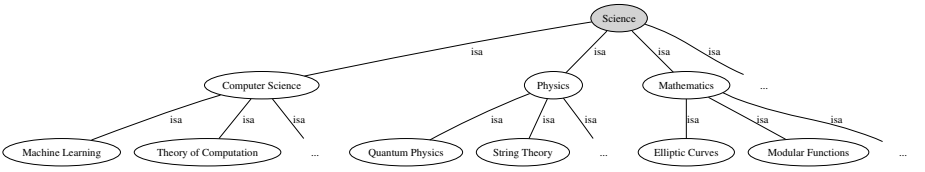


Fig. 1. Illustrative taxonomy of 'Science' by human

Word taxonomies present the possibility of learning classification rules that are simpler and easier-to-understand when the terms in the rules are expressed in terms of abstract values. Kohavi and Provost [5] pointed the need of incorporating hierarchically structured background knowledge. Abstraction of similar concepts by the means of attribute value taxonomy (AVT) has been shown to be useful in generating concise and accurate classifiers [6, 7, 8]. Zhang and Honavar [8] presented AVT-NBL, an algorithm that exploits AVTs to generate Naive Bayes Classifiers that are more compact and often more accurate than classifiers that do not use AVTs. The algorithm potentially performs regularization to minimize over-fitting from learning with relatively small data sets.

Against this background, we introduce word taxonomy guided Naive Bayes learner for the multinomial event model (WTNBL-MN). WTNBL-MN is a word taxonomy based generalization of the standard Naive Bayes learning algorithm for the multinomial model.

Because word taxonomy is not available in many domains, there is a need for automated construction of word taxonomy. Hence, we describe a word taxonomy learner (WTL) that automatically generates word taxonomy from sequence data by clustering of words based on their class conditional distribution.

To evaluate our algorithms, we conducted experiments using two classification tasks: (a) assigning Reuters newswire articles to categories, (b) and classifying protein sequences in terms of their localization. We used Word Taxonomy Learner (WTL) to generate word taxonomy from the training data. The generated word taxonomy was provided to WTNBL-MN to learn concise Naive Bayes classifiers that used abstract words of word taxonomy.

The rest of this paper is organized as follows: Section 2 introduces the WTNBL-MN algorithm; Section 3 presents WTL algorithm; Section 4 describes our experimental results and Section 5 concludes with summary and discussion.

2 Word Taxonomy Guided Naive Bayes Learner for the Multinomial Event Model (WTNBL-MN)

We start with definitions of preliminary concepts necessary to describe our algorithms. We then precisely define the problem as learning classifier from word taxonomy and sequence data.

2.1 Word Taxonomy

Let $\Sigma = \{w_1, w_2, \ldots, w_N\}$ be a dictionary of words, $C = \{c_1, c_2, \ldots, c_M\}$ a finite set of mutually disjoint class labels, and $f_{i,j}$ denote an integer frequency of word w_i in a sequence d_j. Then, sequence d_j is represented as an instance I_j, a frequency vector $< f_{i,j} >$ of w_i, and each sequence belongs to a class label in C. Finally, a data set D is represented as a collection of instance and their associated class label $\{(I_j, c_j)\}$.

Let T_Σ be a word taxonomy defined over the possible words of Σ. Let $Nodes(T_\Sigma)$ denote the set of all values in T_Σ and $Root(T_\Sigma)$ denote the root of T_Σ. We represent the set of leaves of T_Σ as $Leaves(T_\Sigma) \subseteq \Sigma$. The internal nodes of the tree correspond to *abstract values* of Σ.

After Haussler [9], we define a cut γ through a word taxonomy T_Σ as follows.

Definition 1 (Cut). *A cut γ is a subset of nodes in word taxonomy T_Σ satisfying the following two properties:*

1. *For any leaf $l \in Leaves(T_\Sigma)$, either $l \in \gamma$ or l is a descendant of a node in T_Σ.*
2. *For any two nodes f,g $\in \gamma$, f is neither a descendant not an ancestor of g.*

A cut γ induces a partition of words in T_Σ. For example, in figure 1, a cut $\{ComputerScience, Physics, Mathematics\}$ defines a partition over the values of an abstract word 'Science'.

Definition 2 (Refinement). *We say that a cut $\hat{\gamma}$ is a refinement of a cut γ if $\hat{\gamma}$ is obtained by replacing at least one node $v \in \gamma$ by its descendants. Conversely, γ is an abstract of $\hat{\gamma}$*

Figure 2 illustrates a refinement process in word taxonomy T_Σ. The cut $\gamma = \{A, B\}$ is been refined to $\hat{\gamma} = \{A_1, A_2, B\}$ by replacing A with A_1 and A_2. Thus, corresponding hypothesis $h_{\hat{\gamma}}$ is a refinement of h_γ.

Definition 3 (Instance Space). *Any choice of γ defines an input space $\mathscr{I}_\gamma$. If there is a node $\in \gamma$ and $\notin Leaves(T_\Sigma)$, the induced input space $\mathscr{I}_\gamma$ is an abstraction of the original input space $\mathscr{I}$.*

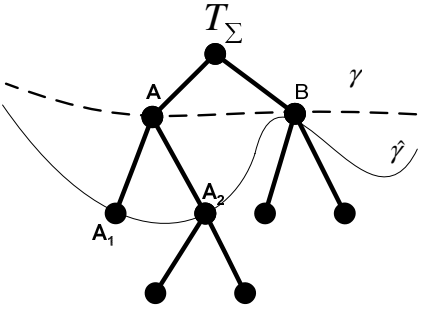


Fig. 2. Illustration of Cut Refinement: The cut $\gamma = \{A, B\}$ is been refined to $\hat{\gamma} = \{A_1, A_2, B\}$ by replacing A with A_1 and A_2

With a data set D, word taxonomy T_Σ and corresponding valid cuts, we can extend our definition of instance space to include instance spaces induced from different levels of abstraction of the original input space. Thus, word taxonomy guided learning algorithm work on this induced input space.

2.2 Event Models for Naive Bayes Sequence Classification

WTNBL-MN algorithm generates a Naive Bayes Classifier for the multinomial model. Before we describe WTNBL-MN algorithm, we briefly summarize event models for Naive Bayes classification of sequence data [10, 11].

Multi-variate Bernoulli Model. In a multi-variate Bernoulli model, a sequence d_j is represented as an instance I_j by a vector of binary values $b_{i,j} \in \{0, 1\}$ where $b_{i,j}$ denotes the presence or absence of a word w_i in the sequence. The number of occurrence of word is not preserved in the vector. The probability of sequence d_j given its class c_j is as follows:

$$P(d_j|c_j) = \prod_{i=1}^{|\Sigma|} (b_{i,j}p_{i,j} + (1 - b_{i,j})(1 - p_{i,j})) \tag{1}$$

Multinomial Model. In a multinomial model, a sequence is represented as a vector of word occurrence frequencies $f_{i,j}$. The probability of an instance I_j given its class c_j is defined as follows:

$$P(d_j|c_j) = \left\{ \frac{\left(\sum_i^{|\Sigma|} f_{i,j}\right)!}{\prod_i^{|\Sigma|}(f_{i,j})!} \right\} \prod_i^{|\Sigma|} \{p_{i,j}^{f_{i,j}}\} \tag{2}$$

The term $\left\{ \frac{\left(\sum_i^{|\Sigma|} f_{i,j}\right)!}{\prod_i^{|\Sigma|}(f_{i,j})!} \right\}$ represents the number of possible combinations of words for the instance I_j.

In equation 2, $p_{i,j}$ is basically calculated as follows:

$$p_{i,j} = \frac{Count(c_j, w_i)}{Count(c_j)}$$

$Count(c_j, w_i)$ is the number of times word w_i appears in all the instances that have a class label c_j, and $Count(c_j)$ is the total number of words in a particular class label c_j. With Laplacian smoothing, $p_{i,j}$ will be as follows:

$$p_{i,j} = \frac{1 + Count(c_j, w_i)}{|\Sigma| + Count(c_j)}$$

2.3 WTNBL-MN Algorithm

The problem of learning classifiers from a word taxonomy and sequence data is a natural generalization of the problem of learning classifiers from the sequence

data. Original data set D is a collection of labeled instances $< I_j, c_j >$ where $I \in \mathscr{I}$. A classifier is a hypothesis in the form of function $h : \mathscr{I} \rightarrow C$, whose domain is the instance space $\mathscr{I}$ and whose range is the set of class C. A hypothesis space $\mathscr{H}$ is a set of hypotheses that can be represented in some hypothesis language or by a parameterized family of functions. Then, the task of learning classifiers from the data set D is induce a hypothesis $h \in \mathscr{H}$ that satisfies given criteria.

Hence, the problem of learning classifiers from word taxonomy and data can be described as follows: Given word taxonomy T_Σ over words Σ and a data set D, the aim is induce a classifier $h_{\gamma^*} : \mathscr{I}_{\gamma^*} \rightarrow C$ where γ^* is a cut that maximizes given criteria. Of interest in this paper is that the resulting hypothesis space $\mathscr{H}_{\hat{\gamma}}$ of a chosen cut $\hat{\gamma}$ is efficient in searching for both concise and accurate hypothesis.

Word taxonomy guided Naive Bayes Learner is composed of two major components: (a) estimation of parameters of Naive Bayes classifiers based on a cut, (b) and a criterion for refining a cut.

Aggregation of Class Conditional Frequency Counts. We can estimate the relevant parameters of a Naive Bayes classifier efficiently by aggregating class conditional frequency counts. For a particular node of a given cut, parameters of the node can be estimated by summing up the class conditional frequency counts of its children [8].

Given word taxonomy T_Σ, we can define a tree of class conditional frequency counts T_f such that there is one-to-one correspondence between the nodes of word taxonomy T_Σ and the nodes of the corresponding T_f. The class conditional frequency counts associated with a non leaf node of T_f is the aggregation of the corresponding class conditional frequency counts associated with its children. Because a cut through word taxonomy corresponds a partition of the set of words, the corresponding cut through T_f specifies a valid class conditional probability table for words. Therefore, to estimate each nodes of T_f, we simply estimate the class conditional frequency counts of primitive words in Σ, which corresponds to the leaves of T_f. Then we aggregate them recursively to calculate the class conditional frequency counts associated with their parent node.

Conditional Minimum Description Length of Naive Bayes Classifier. We use conditional minimum description length (CMDL) [12] score to grade the refinement of Naive Bayes classifier for the multinomial model.

Let v_i^j be the i^{th} attribute value of j^{th} instance $d_j \in D$, and $c_j \in C$ a class label associated with d_j. Then, the conditional log likelihood of the hypothesis B given data D is

$$CLL(B|D) = |D| \sum_{j}^{|D|} \log\{P_B(c_j|d_j)\} = |D| \sum_{j}^{|D|} \log \left\{ \frac{P_B(c_j)P_B(d_j|c_j)}{\sum_{c_k}^{|C|} P_B(c_k)P_B(d_j|c_k)} \right\} \tag{3}$$

For Naive Bayes classifier, this score can be efficiently calculated [8].

$$CLL(B|D) = |D| \sum_{j}^{|D|} \log \left\{ \frac{P(c_j) \prod_i \{P(v_i^j|c_j)\}}{\sum_{c_k}^{|C|} P(c_k) \prod_i \{P(v_i^j|c_k)\}} \right\}$$

And the corresponding conditional minimum description length (CMDL) score is defined as follows:

$$CMDL(B|D) = -CLL(B|D) + \left\{ \frac{\log |D|}{2} \right\} size(B)$$

where, $size(B)$ is a size of the hypothesis B which corresponds to the number of entries in conditional probability tables (CPT) of B.

In case of a Naive Bayes classifier with multi-variate Bernoulli model, $size(B)$ is defined as

$$size(B) = |C| + |C| \sum_{i=1}^{|v|} |v_i|$$

where $|C|$ is the number of class labels, $|v|$ is the number of attributes, and $|v_i|$ is the number of attribute values for an attribute v_i.

Conditional Minimum Description Length of a Naive Bayes Classifier for the Multinomial Model. Combining the equations 2 and 3, we can obtain the conditional log likelihood of the classifier B given data D under the Naive Bayes multinomial model.

$$CLL(B|D) = |D| \sum_{j}^{|D|} \log \left\{ \frac{P(c_j) \left\{ \frac{\left(\sum_i^{|\Sigma|} f_{i,j}\right)!}{\prod_i^{|\Sigma|}(f_{i,j})!} \right\} \prod_i^{|\Sigma|} \{p_{i,j}^{f_{i,j}}\}}{\sum_k^{|C|} \left\{ P(c_k) \left\{ \frac{\left(\sum_i^{|\Sigma|} f_{i,k}\right)!}{\prod_i^{|\Sigma|}(f_{i,k})!} \right\} \prod_i^{|\Sigma|} \{p_{i,k}^{f_{i,k}}\} \right\}} \right\} \quad (4)$$

where, $|D|$ is the number of instances, $c_j \in C$ is a class label for instance $d_j \in D$, $f_{i,j}$ is a integer frequency of word $w_i \in \Sigma$ in instance d_j, and $p_{i,j}$ is the estimated probability that word w_i occurred in the instances associated to class label j.

Conditional Minimum Description Length (CMDL) of a Naive Bayes Classifier for the multinomial model is defined as follows:

$$CMDL(B|D) = -CLL(B|D) + \left\{ \frac{\log |D|}{2} \right\} size(B)$$

where, $size(B)$ is a size of the hypothesis B which corresponds to the number of entries in conditional probability tables (CPT) of B.

Therefore, $size(B)$ is estimated as

$$size(B) = |C| + |C||\Sigma| \quad (5)$$

where $|C|$ is the number of class labels, $|\Sigma|$ is the number of words.

Computation of CMDL Score. Because each word is assumed to be independent of others given the class, the search for the word taxonomy guided Naive Bayes classifier can be performed efficiently by optimizing the CMDL criterion

WTNBL-MN:
begin

1. **Input** : data set D and word taxonomy T_Σ
2. Initialize cut γ to the root of T_Σ
3. Estimate probabilities that specify the hypothesis h_γ
4. Repeat until no change in cut γ
5. $\quad \bar{\gamma} \leftarrow \gamma$
6. $\quad$ For each node $v \in \gamma$:
7. $\quad\quad$ Generate a refinement γ^v of γ by replacing v with its children.
8. $\quad\quad$ Construct corresponding hypothesis h_{γ^v}.
9. $\quad\quad$ If $CMDL(h_{\gamma^v}|D) < CMDL(h_{\bar{\gamma}}|D)$, then replace $\bar{\gamma}$ with γ^v.
10. $\quad$ If $\gamma \neq \bar{\gamma}$ then $\gamma \leftarrow \bar{\gamma}$
11. **Output** : h_γ

end.

Fig. 3. Pseudo-code of Word Taxonomy Guided Naive Bayes Learner for the Multinomial Model(WTNBL-MN)

independently for each word. Thus, the resulting hypothesis h intuitively trades off the complexity in terms of the number of parameters against the accuracy of classification. The algorithm terminates when none of candidate refinements of the classifier yield statistically significant improvement in the CMDL score. Figure 3 outlines the algorithm.

3 Learning a Word Taxonomy from Sequence Data

We describe word taxonomy learner (WTL), a simple algorithm for automated construction of word taxonomy from sequence data.

3.1 Problem Definition

The problem of learning a word taxonomy from sequence data can be stated as follows: Given a data set represented as a set of instances where an instance is a frequency vector $< f_i, c >$ of a word $w_i \in \Sigma$ and associated class label c, and a similarity measure among the words, output word taxonomy T_Σ such that it corresponds to a hierarchical grouping of words in Σ based on the specified similarity measure.

3.2 Algorithm

We use hierarchical agglomerative clustering (HAC) of words based on the distribution of class labels that co-occur with them. Let $DM(P(x)||Q(x))$ denote a measure of pairwise divergence between two probability distributions P and Q of the random variable x.

WTL:
begin

1. **Input** : data set D
2. For each word $w_i \in \Sigma$:
3. For each class $c_k \in C$:
4. Estimate the probability distribution $p(c_k|w_i)$
5. Let $P(C|w_i) = \{p(c_1|w_i), \ldots, p(c_k|w_i)\}$ be the class distribution associated with the word w_i.
6. $\gamma \leftarrow \Sigma$;
7. Initialize T_Σ with nodes in γ.
8. Iterate until $|\gamma| = 1$:
9. In γ, find $(x, y) = argmin\{DM(P(C|x)||P(C|y))\}$
10. Merge x and y $(x \neq y)$ to create a new value z.
11. Calculate probability distribution $P(C|z)$.
12. $\hat{\gamma} \leftarrow \gamma \cup \{z\} \setminus \{x, y\}$.
13. Update T_Σ by adding nodes z as a parent of x and y.
14. $\gamma \leftarrow \hat{\gamma}$.
15. **Output** : T_Σ

end.

Fig. 4. Pseudo-code of Word Taxonomy Learner (WTL)

We use a pairwise measure of divergence between the distribution of the class labels associated with the corresponding words as a measure of dissimilarity between the words. The lower the divergence between the class distribution between two words, the greater is their similarity. The choice of this measure of dissimilarity is motivated by the intended use of word taxonomy for WTNBL-MN algorithm to generate concise and accurate classifiers. If two words are indistinguishable from each other with respect to their class distribution, they will provide statistically similar information for classification of instance.

The pseudocode for the Word Taxonomy Learner (WTL) is shown in figure 4. The basic idea is to construct a taxonomy T_Σ by starting with the primitive words in Σ as the leaves of T_Σ and recursively add nodes to T_Σ one at a time by merging two existing nodes. To aid this process, the algorithm maintains a cut γ through the taxonomy T_Σ, updating the cut γ as new nodes are added to T_Σ. At each step, the two words to be grouped together to obtain an abstract word to be added to T_Σ are selected from γ based on the divergence between the class distributions associated with the corresponding words. That is, a pair of words in γ are merged if they have more similar class distributions than any other pair of words in γ. This process terminates when the cut γ contains a single word which corresponds to the root of T_Σ. The resulting T_Σ will have $(2|\Sigma| - 1)$ nodes when the algorithm terminates.

3.3 Dissimilarity Measure

Several ways of measuring dissimilarity between two probability distributions have been studied in the literature [13]. We have experimented with thirteen different divergence measures. In our experiments, most of them resulted in similar performance on classification tasks. Hence, we focus on the results obtained by Jensen-Shannon divergence measure in the discussion that follows [14].

Jensen Shannon Divergence. is a weighted information gain that is reflexive, symmetric and bounded. Pairwise version of Jensen-Shannon divergence is given by

$$I\left(P||Q\right) = \frac{1}{2}\left[\sum p_i log\left(\frac{2p_i}{p_i+q_i}\right) + \sum q_i log\left(\frac{2q_i}{p_i+q_i}\right)\right]$$

4 Experiments

The results of experiments described in this section provide evidence that WTNBL-MN coupled with WTL usually generate more concise and often more accurate classifiers than those of the Naive Bayes classifiers for the multinomial model. We conducted experiments with two sequence classification tasks; text (word sequence) classification and proteins (amino acid sequence) classification. In each case, a word taxonomy is generated using WTL and a classifier is constructed using WTNBL-MN on the resulting WT and sequence data.

4.1 Text Classification

Reuters 21587 distribution 1.0 data set[1] consists of 12902 newswire articles in 135 overlapping topic categories.

We build binary classifiers for top ten most populous categories on text classification [15, 16, 10]. In our experiment, stop words were not eliminated, and title words were not distinguished with body words. We selected top 300 features based on mutual information with class labels. The mutual information $MI(x, c)$ between a feature x and a category c is defined as:

$$MI(x,c) = \sum^{x}\left\{\sum^{c}\left\{P(x,c)log\frac{P(x,c)}{P(x)P(c)}\right\}\right\}$$

We followed the ModApte split [17] in which 9603 stories are used for building classifiers and 3299 stories to test the accuracy of the resulting model. We report the break even points, the average of precision and recall when the difference between the two is minimum. Precision and recall of text categorization are defined as:

[1] This collection is publicly available at http://www.daviddlewis.com/resources/testcollections/reuters21578/.

$$\text{Precision} = \frac{|\text{detected documents in the category (true positives)}|}{|\text{documents in the category (true positives + false negatives)}|}$$

$$\text{Recall} = \frac{|\text{detected documents in the category (true positives)}|}{|\text{detected documents (true positives + false positives)}|}$$

Table 1 shows the break even point of precision and recall as well as the size of the classifier (from the equation 5) for the ten most frequent categories. WTNBL-MN usually shows similar performance in terms of break even performance except in the case of "corn" category, while the classifiers generated by WTNBL-MN have smaller size than those generated by the Naive Bayes Learner (NBL).

Table 1. Break even point of 10 Largest Categories

Data	NBL-MN		WTNBL-MN		# of documents	
	breakeven	size	breakeven	size	train	test
earn	94.94	602	94.57	348	2877	1087
acq	89.43	602	89.43	472	1650	719
money-fx	64.80	602	65.36	346	538	179
grain	74.50	602	77.85	198	433	149
crude	79.89	602	76.72	182	389	189
trade	59.83	602	47.01	208	369	118
interest	61.07	602	59.54	366	347	131
ship	82.02	602	82.02	348	197	89
wheat	57.75	602	53.52	226	212	71
corn	57.14	602	21.43	106	182	56

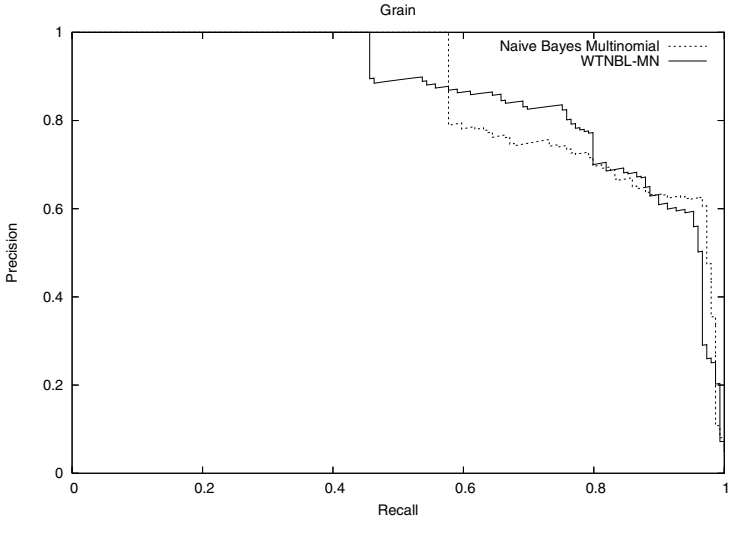


Fig. 5. Precision-Recall Curves of "Grain" Category

Figure 5 shows Precision-Recall curve [18] for the "grain" category. It can be seen that WTNBL-MN generates a Naive Bayes classifier that is more compact than, but has performance comparable to that of the classifier generated from Naive Bayes learner.

WTNBL-MN did not show good performance for the "corn" category. This may be explained by the fact that conditional minimum description length trades off the accuracy of the model against its complexity, which may not necessarily optimize precision & recall for a particular class As a consequence, WTNBL-MN may terminate refinement of the classifier prematurely for class labels with low support, i.e. when the data set is imbalanced.

4.2 Protein Sequence Classification

We applied WTNBL-MN algorithm on two protein data sets with a view to identifying their localization [19].

The first data set is 997 prokaryotic protein sequences derived from SWISS-PROT data base [20]. This data set includes proteins from three different subcellular locations: cytoplasmic (688 proteins), periplasmic (202 proteins), and extracellular (107 proteins).

The second data set is 2427 eukaryotic protein sequences derived from SWISS-PROT data base [20]. This data set includes proteins from the following four different subcellular locations: nuclear (1097 proteins), cytoplasmic (684 proteins), mitochondrial (321 proteins), extracellular (325 proteins).

For these data sets[2], we conducted ten-fold cross validation. To measure the performance of the following performance measures [21] are applied and the results for the data set are reported:

$$\text{Correlation coefficient} = \frac{\text{TP} \times \text{TN} - \text{FP} \times \text{FN}}{\sqrt{(\text{TP+FN})(\text{TP+FP})(\text{TN+FP})(\text{TN+FN})}}$$

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP+TN+FP+FN}}$$

$$\text{Sensitivity}^{+} = \frac{\text{TP}}{\text{TP+FN}}$$

$$\text{Specificity}^{+} = \frac{\text{TP}}{\text{TP+FP}}$$

where, TP is the number of true positives, FP is the number of false positives, TN is the number of true negatives, and FN is the number of false negatives.

Figure 6 is amino acid taxonomy constructed for the prokaryotic protein sequences. Table 2 shows the results in terms of the performance measures for the two protein sequences. For both data sets, the classifier generated by WTNBL is more concise and shows more accurate performance than the classifier generated by the Naive Bayes Learner (NBL) in terms of the measures reported.

[2] These datasets are available to download at http://www.doe-mbi.ucla.edu/~astrid/astrid.html.

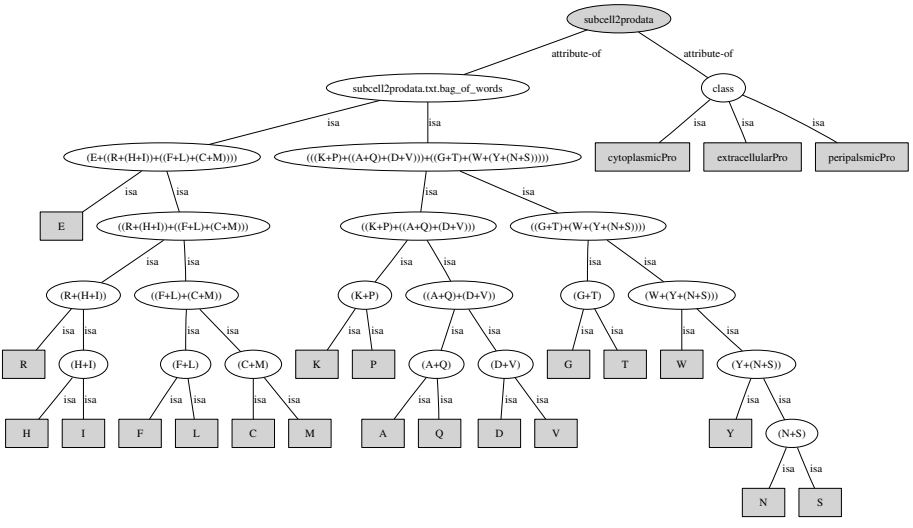


Fig. 6. Taxonomy from Prokaryotic Protein Localization Sequences constructed by WTL

Table 2. Results on Protein Localization Sequences (abbrev.: C - cytoplasmic, E - extracellular, P - peripalsmic, N - nuclear, M - mitochondrial)

Method	Prokaryotic			Eukaryotic			
	C	E	P	N	E	M	C
NBL-MN correlation coefficient	71.96	70.57	51.31	61.00	36.83	25.13	44.05
accuracy	88.26	93.58	81.85	80.72	83.11	71.69	71.41
specificity$^+$	89.60	65.93	53.85	82.06	40.23	25.85	49.55
sensitivity$^+$	93.90	83.18	72.77	73.38	53.85	61.06	81.29
size	42	42	42	46	46	46	46
WTNBL-MN correlation coefficient	72.43	69.31	51.53	60.82	38.21	25.48	43.46
accuracy	88.47	93.18	81.85	80.63	84.01	72.35	71.24
specificity$^+$	89.63	64.03	53.82	81.70	42.30	26.29	49.37
sensitivity$^+$	94.19	83.18	73.27	73.66	53.23	60.44	80.56
size	20	20	40	24	36	34	32

5 Summary and Related Work

5.1 Summary

We have presented word taxonomy guided Naive Bayes Learning algorithm for the multinomial event model (WTNBL-MN). We also described WTL, an algo-

rithm for automated generation of word taxonomy for sequence data. WTNBL-MN is a generalization of the Naive Bayes learner for the multinomial event model for learning classifiers from data using word taxonomy. WTL is a hierarchical agglomerative clustering algorithm to cluster words into taxonomy based on the distribution of class labels that co-occur with the words. Experimental results on protein sequence and Reuters text show that the proposed algorithms can generate Naive Bayes classifiers that are more compact and often more accurate than those produced by standard Naive Bayes learner for the Multinomial Model.

5.2 Related Work

Several groups have explored the problem of learning classifiers from attribute value taxonomies (AVT) or tree structured attributes: Zhang and Honavar [6, 8] developed decision tree learner and Naive Bayes learner regularized over attribute value taxonomy. These works were primarily focused on attribute value taxonomy for multi-variate data sets. Taylor et al. [22] and Hendler et al. [23] described the use of taxonomy in rule learning. Han and Fu [24] proposed a method for exploiting hierarchically structured background knowledge for learning association rules. desJardins et al. [25] suggested the use of Abstraction-Based-Search (ABS) to learning Bayesian networks with compact structure.

Gibson and Kleinberg [26] introduced STIRR, an iterative algorithm based on non-linear dynamic systems for clustering categorical attributes. Ganti et. al. [27] designed CACTUS, an algorithm that uses intra-attribute summaries to cluster attribute values. Both of them did not make taxonomies and use the generated for improving classification tasks.

Pereira et. al. [28] described distributional clustering for grouping words based on class distributions associated with the words in text classification. Slonim and Tishby [14] described a technique (called the agglomerative information bottleneck method) which extended the distributional clustering approach described by Pereira et al. [28], using Jensen-Shannon divergence for measuring distance between document class distributions associated with words and applied it to a text classification task. Baker and McCallum [29] reported improved performance on text classification using a distributional clustering with a Jensen-Shannon divergence measure.

To the best of our knowledge, the results presented in this paper are the first of these kinds with regards to exploitation of word taxonomies in the generation of compact yet accurate classifiers for sequence classification.

5.3 Future Work

Some promising directions for future work include the following:

- Application of the WTNBL-MN algorithm to up-to-date text corpora [30, 31].
- Enhancement of the WTNBL-MN and WTL algorithms for learning and exploiting hierarchical ontologies based on part-whole and other relations as opposed to ISA relations.

– Development of other measures for model selection rather than CMDL for cut refinement to accommodate the various application-specific needs.

References

1. Pazzani, M.J., Mani, S., Shankle, W.R.: Beyond concise and colorful: Learning intelligible rules. In: Knowledge Discovery and Data Mining. (1997) 235–238
2. Ashburner, M., Ball, C., Blake, J., Botstein, D., Butler, H., Cherry, J., Davis, A., Dolinski, K., Dwight, S., Eppig, J., Harris, M., Hill, D., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J., Richardson, J., Ringwald, M., Rubin, G., Sherlock, G.: Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. Nature Genetics **25** (2000) 25–29
3. Undercoffer, J.L., Joshi, A., Finin, T., Pinkston, J.: A Target Centric Ontology for Intrusion Detection: Using DAML+OIL to Classify Intrusive Behaviors. Knowledge Engineering Review (2004)
4. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American (2001)
5. Kohavi, R., Provost, F.: Applications of data mining to electronic commerce. Data Mining and Knowledge Discovery **5** (2001) 5–10
6. Zhang, J., Honavar, V.: Learning decision tree classifiers from attribute value taxonomies and partially specified data. In: the Twentieth International Conference on Machine Learning (ICML 2003), Washington, DC (2003)
7. Kang, D.K., Silvescu, A., Zhang, J., Honavar, V.: Generation of attribute value taxonomies from data for data-driven construction of accurate and compact classifiers. In: Proceedings of the 4th IEEE International Conference on Data Mining (ICDM 2004), 1-4 November 2004, Brighton, UK. (2004) 130–137
8. Zhang, J., Honavar, V.: AVT-NBL: An algorithm for learning compact and accurate naive bayes classifiers from attribute value taxonomies and data. In: International Conference on Data Mining (ICDM 2004). (2004)
9. Haussler, D.: Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. Artificial intelligence **36** (1988) 177 – 221
10. McCallum, A., Nigam, K.: A comparison of event models for naive bayes text classification. In: AAAI-98 Workshop on Learning for Text Categorization. (1998)
11. Eyheramendy, S., Lewis, D.D., Madigan, D.: On the naive bayes model for text categorization. In: Ninth International Workshop on Artificial Intelligence and Statistics. (2003)
12. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian network classifiers. Mach. Learn. **29** (1997) 131–163
13. Arndt, C.: Information Measures. Springer-Verlag Telos (2001)
14. Slonim, N., Tishby, N.: Agglomerative information bottleneck. In: Proceedings of the 13th Neural Information Processing Systems (NIPS 1999). (1999)
15. Dumais, S., Platt, J., Heckerman, D., Sahami, M.: Inductive learning algorithms and representations for text categorization. In: CIKM '98: Proceedings of the seventh international conference on Information and knowledge management, ACM Press (1998) 148–155
16. Joachims, T.: Text categorization with support vector machines: learning with many relevant features. In Nédellec, C., Rouveirol, C., eds.: Proceedings of ECML-98, 10th European Conference on Machine Learning, Chemnitz, DE, Springer Verlag, Heidelberg, DE (1998) 137–142

17. Apté, C., Damerau, F., Weiss, S.M.: Towards language independent automated learning of text categorization models. In: SIGIR '94: Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval, New York, NY, USA, Springer-Verlag New York, Inc. (1994) 23–30
18. Fawcett, T.: ROC graphs: Notes and practical considerations for researchers. Technical Report HPL-2003-4, HP Labs (2003)
19. Andorf, C., Silvescu, A., Dobbs, D., Honavar, V.: Learning classifiers for assigning protein sequences to gene ontology functional families. In: Fifth International Conference on Knowledge Based Computer Systems (KBCS 2004). (2004) 256–265
20. Bairoch, A., Apweiler, R.: The SWISS-PROT protein sequence database and its supplement TrEMBL in 2000. Nucleic Acids Res. **28** (2000) 45–48
21. Yan, C., Dobbs, D., Honavar, V.: A two-stage classifier for identification of protein-protein interface residues. In: Proceedings Twelfth International Conference on Intelligent Systems for Molecular Biology / Third European Conference on Computational Biology (ISMB/ECCB 2004). (2004) 371–378
22. Taylor, M.G., Stoffel, K., Hendler, J.A.: Ontology-based induction of high level classification rules. In: DMKD. (1997)
23. Hendler, J., Stoffel, K., Taylor, M.: Advances in high performance knowledge representation. Technical Report CS-TR-3672, University of Maryland Institute for Advanced Computer Studies Dept. of Computer Science (1996)
24. Han, J., Fu, Y.: Exploration of the power of attribute-oriented induction in data mining. In Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R., eds.: Advances in Knowledge Discovery and Data Mining. AIII Press/MIT Press (1996)
25. desJardins, M., Getoor, L., Koller, D.: Using feature hierarchies in bayesian network learning. In: SARA '02: Proceedings of the 4th International Symposium on Abstraction, Reformulation, and Approximation, Springer-Verlag (2000) 260–270
26. Gibson, D., Kleinberg, J.M., Raghavan, P.: Clustering categorical data: An approach based on dynamical systems. VLDB Journal: Very Large Data Bases **8** (2000) 222–236
27. Ganti, V., Gehrke, J., Ramakrishnan, R.: Cactus - clustering categorical data using summaries. In: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM Press (1999) 73–83
28. Pereira, F., Tishby, N., Lee, L.: Distributional clustering of English words. In: 31st Annual Meeting of the ACL. (1993) 183–190
29. Baker, L.D., McCallum, A.K.: Distributional clustering of words for text classification. In: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval, ACM Press (1998) 96–103
30. Lewis, D.D., Yang, Y., Rose, T.G., Li, F.: RCV1: A new benchmark collection for text categorization research. J. Mach. Learn. Res. **5** (2004) 361–397
31. Klimt, B., Yang, Y.: The Enron corpus: A new dataset for email classification research. In: 15th European Conference on Machine Learning (ECML2004). Vol. 3201 of Lecture Notes in Computer Science : Springer-Verlag. (2004) 217–226

Petri Net Reachability Checking Is Polynomial with Optimal Abstraction Hierarchies

Peep Küngas

Norwegian University of Science and Technology,
Department of Computer and Information Science, Trondheim, Norway
peep@idi.ntnu.no

Abstract. The Petri net model is a powerful state transition oriented model to analyse, model and evaluate asynchronous and concurrent systems. However, like other state transition models, it encounters the state explosion problem. The size of the state space increases exponentially with the system complexity.

This paper is concerned with a method of abstracting automatically Petri nets to simpler representations, which are ordered with respect to their size. Thus it becomes possible to check Petri net reachability incrementally. With incremental approach we can overcome the exponential nature of Petri net reachability checking. We show that by using the incremental approach, the upper computational complexity bound for Petri net reachability checking with optimal abstraction hierarchies is polynomial.

The method we propose considers structural properties of a Petri net as well an initial and a final marking. In addition to Petri net abstraction irrelevant transitions for a given reachability problem are determined. By removing these transitions from a net, impact of the state explosion problem is reduced even more.

1 Introduction

Petri nets and related graph models have been proposed for a wide variety of applications. These models are particularly suitable for representing concurrent hardware and software systems. A fundamental basis for studying the dynamic properties of systems described with Petri nets is the reachability property [11].

Unfortunately the complexity of Petri net reachability checking has been proven to be EXPSPACE-hard [10] in the general case. Although several less complex classes of nets have been determined [4], there still are problems, which can be presented only with "unconstrained" Petri nets. Therefore tools and algorithms for coping with that EXPSPACE-hard complexity are urgently needed.

One possible way to reduce the state space is to apply net abstraction techniques in conjunction with Petri net reachability checking. Abstraction techniques have been used extensively in Artificial Intelligence (AI) planning [1, 3, 9] (especially in case-based and analogical reasoning), human problem solving and

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 149–164, 2005.

(automated) theorem proving [12]. Korf [8] has shown that while using abstraction it is possible to reduce the expected search space from $O(n)$ to $O(log\ n)$. This improvement makes combinatorial problems tractable. For instance, if n is an exponential function of problem size, then $log\ n$ is just linear (according to [8]).

The essential reason why abstraction reduces complexity is that the total complexity is the sum of complexities of multiple searches, not their product. Thus with abstraction techniques we may cut solution search space from b^d to $kb^{d/k}$, where b is the branching factor and d is the depth of the search space while k is the ratio of abstraction space to base space.

Although abstraction of Petri nets has been already explored for instance in [14, 13, 15], the proposed approaches are based on analysing merely structural properties of nets though in some cases also initial markings have been taken into account.

The methodology we propose, on the contrary, considers structural properties as well the initial and the final marking of a Petri net. While preparing the net for reachability checking we use the final marking to determine transitions not relevant for the given reachability problem. Thereby it becomes possible to remove some transitions from a net, without changing the reachability result for the given final marking. In that way more efficient reachability checking could be implemented.

The initial marking helps us to recognise the negative answer to a reachability problem. Thus, in some cases we can determine whether the final marking is reachable from the initial one even before we start with reachability checking. Additionally we present a methodology for abstraction-based reachability checking and prove that if a Petri net has an optimal abstraction hierarchy, then the computational complexity of its reachability checking is polynomial.

The remainder of the paper is as follows. In Section 2 we present the main definitions used in the rest of the paper. Section 3 introduces the abstraction algorithm and analyses its influence to Petri nets and their properties. In Section 4 reachability checking with abstraction is described. This methodology is evaluated in Section 5, where experimental results are presented and analysed. Section 6 gives an overview of related work and finally in Section 7 everything is summed up.

2 Definitions

In this section we define the Petri net concept and give the main notation and definitions to be used in the sequel.

A (marked) Petri net is a 5-tuple $N = (P, T, Pre, Post, M_0)$, where $P = \{p_1, p_2, \ldots, p_n\}$ is a finite set of places, $T = \{t_1, t_2, \ldots, t_m\}$ is a finite set of transitions, $Pre : P \times T \to \mathcal{N}$ is the input incidence function, $Post : T \times P \to \mathcal{N}$ is the output incidence function and $M_0 : P \to \mathcal{N}$ is the initial marking. The number of places and transitions in a Petri net are represented with $|P|$ and $|T|$ respectively. The size $\mathcal{S}$ of a Petri net is the number of places it involves—$\mathcal{S}(N) = |P|$.

In the graphical representation, circles denote places and vertical bars denote transitions, tokens are represented with dots inside places. The *Pre* incidence function describes the oriented arcs connecting places to transitions. It represents for each transition t the fragment of the state in which the system has to be before the state change corresponding to t occurs. $Pre(p,t)$ is the weight of the arc (p,t), $Pre(p,t) = 0$ denotes that the place p is not connected to transition t.

The *Post* incidence function describes arcs from transitions to places. Analogously to *Pre*, $Post(t,p)$ is the weight of the arc (t,p). The vectors $Pre(.,t)$ and $Post(t,.)$ denote respectively all input and output arcs of transition t with their weights.

The Petri net dynamics is given by firing enabled transitions, whose occurrence corresponds to a state change of the system modeled by the net. A transition t is enabled for a marking M, if $M \geq Pre(.,t)$. This enabling condition is equivalent to $\forall p \in P, M(p) \geq Pre(p,t)$. Only enabled transitions can be fired.

If M is a marking of N enabling a transition t, and M' is the marking derived by the firing of a transition t from M, then $M' = (M - Pre(.,t)) + Post(t,.)$. The firing is denoted as $M \xrightarrow{t} M'$. Firing of a sequence of transitions $s = \langle t_1 \dots t_n \rangle$ is defined as $M_0 \xrightarrow{s} M_n = M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \dots \xrightarrow{t_n} M_n$, where $M_i, i = 0 \dots n-1$ is a marking of N enabling a transition t_{i+1} and M_{i+1} is a result of firing t_{i+1} from marking M_i.

In a Petri net N it is said that a marking M_g is reachable from a marking M iff there exists a sequence of transitions s such that $M \xrightarrow{s} M_g$. We call the *reachability problem* for Petri nets the problem of finding a firing sequence s to reach a given marking M_g from the initial marking M_0. The *coverability problem* (sometimes also called the submarking reachability problem), given two markings M_0 and M_g, is defined as the problem of finding a firing sequence s to reach a marking M_s from the initial marking M_0 such that $M_g \leq M_s$. To simplify representing a marking M, we use symbolic representation $\mathcal{M}(M)$, defined in the following way: $\mathcal{M}(M) = \{p^{M(p)} \mid p \in P, M(p) \geq 1\}$. We shall write p instead of p^1.

Abstraction hierarchy $\mathcal{H}$ for a Petri net N is a total order of abstractions such that $\forall \mathcal{A}_i, \mathcal{A}_j \in \mathcal{H}, i \neq j, (\mathcal{S}(\mathcal{A}_i(N)) < \mathcal{S}(\mathcal{A}_j(N))) \Rightarrow \mathcal{A}_j(N) \prec \mathcal{A}_i(N)$. Due to the way we construct abstraction levels, it is not possible that $\mathcal{S}(\mathcal{A}_i(N)) \equiv \mathcal{S}(\mathcal{A}_j(N))$, if $i \neq j$.

If p is a Petri net place, then $Level(p)$ is the highest abstraction level where p may appear. To explain this notion let us consider Figure 4. There $Level(X) \equiv 2$, meaning that place X occurs in all abstracted versions of a net starting from level 2. Similarily $Level(F) \equiv 1$ and $Level(M) \equiv 0$ on the same figure.

The function $\mathcal{E}(M) = \{p \mid p \in P, M(p) > 0\}$ returns a set consisting of nonempty places according to a marking M. A Petri net marking M at abstraction level i is denoted by $\mathcal{A}_i(M)$, where

$$\mathcal{A}_i(M)(p) = \begin{cases} M(p) & \text{if } Level(p) \geq i \\ 0 & \text{otherwise} \end{cases}$$

Abstracted Petri net N at abstraction level i is defined as

$$\mathcal{A}_i(N) = (P', T', Pre, Post, M_0'),$$

where:

- $P' = \{p \mid p \in P, Level(p) \geq i\}$
- $T' = \{t \mid t \in T, \mathcal{E}(\mathcal{A}_i(Pre(.,t))) \cup \mathcal{E}(\mathcal{A}_i(Post(t,.))) \neq \emptyset\}$
- $M_0' = \mathcal{A}_i(M_0)$

At abstraction level 0 an original Petri net is presented—$\mathcal{A}_0(N) \equiv N$. We write T_i and P_i to denote respectively a set of transitions and places of a Petri net $\mathcal{A}_i(N)$.

Abstracted sequence of Petri net transitions $s = \langle t_1, t_2, \ldots, t_n \rangle, t_j \in T, j = 1 \ldots n$ at abstraction level i is defined as $\mathcal{A}_i(s) = \langle s_j \mid 0 < j \leq |s|, s_j \in T_i \rangle$. Basically it means that at abstraction level i in a sequence s only these transitions are allowed, which exist in the abstracted Petri net at abstraction level i. Other transitions are just discarded.

Opposite operation to abstration of firing sequences is refinement. Refinement $\mathcal{R}_k^l, k > l$ of a firing sequence $s = \langle t_1, t_2, \ldots, t_n \rangle, t_j \in T, j = 1 \ldots n$ from abstraction level k to abstraction level l is defined as a sequence $\mathcal{R}_k^l(s) = \langle \alpha_0, t_1, \alpha_1, \ldots \alpha_{n-1}, t_n, \alpha_n \rangle$, where $\alpha_i, i = 0 \ldots n$ is a sequence of transitions from $t \in \bigcup_{i=k-1}^{l} T_i$. This means that during refinement only transitions from lower abstraction levels may be inserted to firing sequences. We write $\mathcal{R}^j$ instead of $\mathcal{R}_{j+1}^j$.

We define a nulltransition as a transition t such that $\mathcal{E}(Pre(.,t)) \equiv \emptyset \wedge \mathcal{E}(Post(t,.)) \equiv \emptyset$. Source transition is a transition t such that $\mathcal{E}(Pre(.,t)) \equiv \emptyset \wedge \mathcal{E}(Post(t,.)) \not\equiv \emptyset$. In the following we shall write level to denote an abstraction level and by net we mean a Petri net.

3 Automatic Abstraction of Petri Nets

In this section we describe how to construct abstraction hierarchies for Petri nets. These hierarchies are later used to gradually refine an abstract solution during reachability checking. The abstraction method, we propose here for Petri nets, has been inspired from an abstraction method [7] from the field of AI planning.

The significant difference of our method from previous Petri net abstraction methods is that net fragments are not replaced with more abstract nets. Instead we simplify nets by removing places which are not so relevant at particular abstraction levels and nulltransitions, which were formed by removing specific places from a net representation.

3.1 The Abstraction Algorithm

Given a problem space, which consists of a Petri net and a reachability problem (finding a sequence s so that $M_0 \xrightarrow{s} M_g$), our algorithm reformulates the original

problem into more abstract ones. These more abstract ones are organised into an ordered hierarchy $\mathcal{H}$ with respect to their size. Thus the smallest representation is presented at the highest abstraction level and the largest respectively at the lowest (0) level. The original problem is mentioned in the further text also as the base problem, because all other representations are based on it.

The ordered monotonicity property is used as the basis for generating abstraction hierarchies. This property captures the idea that if an abstract solution is refined, the structure of the abstract solution should be maintained. Hence elements in a transition firing sequence s_i, such that $\mathcal{A}_i(M_0) \xrightarrow{s_i} \mathcal{A}_i(M_g)$, would not be reordered while extending this sequence at abstraction level $i-1$. The process of refining an abstract solution requires inserting additional transitions to achieve the tokens ignored at more abstract level.

Definition 1. *Ordered monotonic refinement $\mathcal{R}$ is a refinement of an abstract solution s so that $\mathcal{A}_i(\mathcal{R}_i^j(s)) = s, j \leq i$, where s is a sequence of transitions, i denotes the abstraction level, where solution s was found and j is the target abstraction level.*

Definition 2. *Ordered monotonic hierarchy is an abstraction hierarchy with the property that for every solvable problem there exists an abstract solution that has a sequence of ordered monotonic refinements into the base space.*

Definition 3. *Let A and B be arbitrary vertices in a directed graph. Then we say that they are strongly connected, if there exists a cycle with A as its initial and final vertice such that this cycle includes vertice B.*

An ordered monotonic abstraction hierarchy is constructed by dividing places P in a Petri net N between abstraction levels such that the places at level i do not interact with any places at level $i+1$. We say that places A and B do not interact with each other, if they are not strongly connected in the dependency graph of the particular Petri net.

Our abstraction algorithm first generates a graph representing dependencies (see Figure 1) between places in a Petri net, and then, using that graph, finally generates abstraction hierarchies (see Figure 2). The algorithm in Figure 1 considers every non-empty place, which has not yet been considered, from a given marking. Then all transitions, which increase the number of tokens in that place, are selected. Connections from the place under consideration to places affected by these transitions are created and search follows recursively, using $Pre(.,t)$ as a new marking, until all reachable places have been processes. It has to be noted that although the algorithm starts from the goal marking M_g, weights of transition input arcs are considered when it proceeds recursively.

After the dependency graph has been constructed, additional connections are created from places $p \in \mathcal{E}(M_g)$ to places $\mathcal{E}(Pre(.,t))$ of transitions t, if $\mathcal{E}(Post(t,.)) \equiv \emptyset$. The latter is due to the fact that the dependency graph is extended by observing arcs $(t,.)$ and, if $\mathcal{E}(Post(t,.)) \equiv \emptyset$, transition t is ignored. Anyway, these transitions may be needed during reachability checking.

```
Algorithm DetermineConstraints(graph, net, marking)

inputs: a Petri net and a final marking
output: constraints, which guarantee ordered monotonicity for a given marking

begin
for ∀place ∈ E(marking)//a nonempty place is selected
    if not(ConstraintsDetermined(place, graph)) then
        ConstraintsDetermined(place, graph) ← true
        for ∀t ∈ net.T//for all transitions
            if place ∈ E(net.Post(t, .)) then
                for ∀p ∈ E(net.Post(t, .))
                    AddDirectedEdge(place, p, graph)
                end for
                for ∀p ∈ E(net.Pre(., t))
                    AddDirectedEdge(place, p, graph)
                end for
                DetermineConstraints(graph, net, net.Pre(., t))
            end if
        end for
    end if
end for
return graph
end DetermineConstraints
```

Fig. 1. Building a dependency graph

```
Algorithm CreateHierarchy(net, marking)

inputs: a Petri net and a final marking
output: an ordered monotonic abstraction hierarchy

begin
graph ← DetermineConstraints({}, net, marking)
components ← FindStronglyConnectedComponents(graph)
partialOrder ← ConstructReducedGraph(graph, components)
absHierarchy ← TopologicalSort(partialOrder)
return absHierarchy
end CreateHierarchy
```

Fig. 2. Creating an abstraction hierarchy

To demonstrate the construction of abstraction hierarchies, let us consider the Petri net in Figure 4(a), where the reachability of marking M_g, $\mathcal{M}(M_g) = \{E, M, X^2\}$ is considered. The algorithm in Figure 1 starts with place E and sets it to be determined. The only transition having E in its outputs is H_EIM.

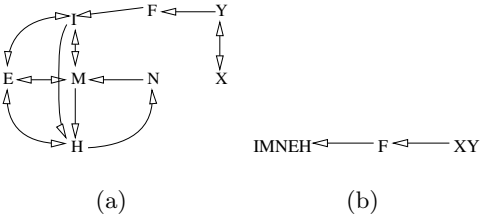


Fig. 3. A dependency graph (a) and strongly connected component groups (b) for a Petri net in Figure 4(a)

Thus edges $E \rightarrow I$, $E \rightarrow M$ and $E \rightarrow H$ are constructed. Then the algorithm proceeds with new marking $\{H\}$ at recursion level 2. H is set to be determined. The only transition having H in its output is *EN_H* and thus edges $H \rightarrow E$ and $H \rightarrow N$ are constructed.

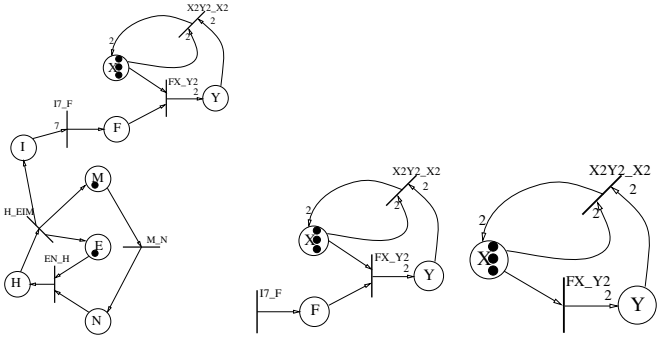
The algorithm proceeds at recursion level 3 with marking $\{E, N\}$. Since E was already set to be determined, it is not considered anymore. Thus N is selected. Transition *M_N* has N in its outputs and thus edge $N \rightarrow M$ is constructed. The algorithm proceeds recursively at level 4 with $\{M\}$ as its marking. M is set to be determined, transition *H_EIM* is the only transition having M in its output and therefore edges $M \rightarrow I$, $M \rightarrow E$ and $M \rightarrow H$ are constructed.

The algorithm proceeds with marking $\{H\}$ at recursion level 5. However, H has already been processed and is not considered anymore. Since there is nothing to do in previous recursion levels, the algorithm returns to level 1 and chooses X for its next target. X is set to be determined, transition *X2Y2_X2* is chosen and edge $X \rightarrow Y$ is inserted. The algorithm proceeds at recursion level 2 with $\{X, Y\}$ as an input marking. Y is chosen, transition *FX_Y2* is found and edges $Y \rightarrow F$ and $Y \rightarrow X$ are inserted.

Then the algorithm proceeds at recursion level 3 witn $\{F, X\}$ as its input marking. Since F is selected, transition *I7_F* is found and edge $F \rightarrow I$ is inserted. Now the algorithm proceeds with marking $\{I\}$ at recursion level 4. Transition *H_EIM* is selected and edges $I \rightarrow E$, $I \rightarrow M$ plus $I \rightarrow H$ are generated. Finally the algorithm proceeds at recursion level 5 with $\{H\}$ as its input marking. However, since there are no nondetermined places left, the algorithm returns to its initial recursion level and returns a graph containing all the edges pointed out so far.

After applying the algorithm in Figure 1, first a dependency graph in Figure 3(a) is generated. In that figure for the sake of simplicity we presented every two unidirectional edges in opposite directions between the same pair of elements as a bidirectional edge. A directed edge from node A to node B in the graph indicates that A cannot occur lower in the abstraction hierarchy than B. Hence a bidirectional edge between two nodes tells us that these nodes should appear at the same abstraction level.

Given the algorithm in Figure 2 we find as a second step strongly connected components. In that way we end up with 3 sets of strongly connected



(a) $\mathcal{A}_0(N) \equiv N$. (b) $\mathcal{A}_1(N)$. (c) $\mathcal{A}_2(N)$.

Fig. 4. Representation of the Petri net N at different abstraction levels with $\mathcal{M}(M_g) = \{E, M, X^2\}$ and $\mathcal{M}(M_0) = \{E, M, X^3\}$

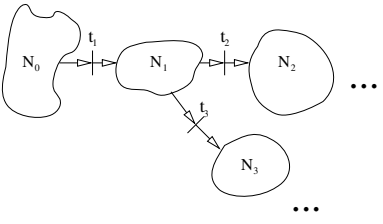


Fig. 5. The general structure of abstracted nets

components—$\{I, M, N, E, H\}$, $\{F\}$ and $\{X, Y\}$. By representing every group of strongly connected components as a separate node, we end up with a partial ordering in Figure 3(b). In the current example it is also a total order and would represent an achieved abstraction hierarchy. Otherwise we would have to apply topological sort to generate all possible total orderings and then select between those a suitable one.

The abstraction hierarchy in Figure 3(b) determines that at the lowest abstraction level all places are presented, at the first level all except I, M, N, E, H (thus F, X, Y) are presented. And at the second abstraction level there are only X and Y. Given this abstraction hierarchy, two new Petri nets in Figure 4(b) and Figure 4(c) are constructed for abstraction levels 1 and 2, respectively. At abstraction level 0 the original Petri net is represented.

Basically, the abstraction algorithm in Figure 2 divides Petri nets into subnets to fit the general tree structure presented in Figure 5. Each subnet N_i there consists of new transitions at abstraction level i. Every leaf subnet of that tree involves at least one place $p \in \mathcal{E}(M_g)$. There can be several transitions between different subnets—every t_i in Figure 5 represents a number (at least one) of transitions between two subnets. However, the direction of transitions in set t_i is the same—from lower level to higher level net. Transitions t_i are abstracted

to source transitions at level i, but at abstraction level $i-1$ they are not source transitions anymore.

This structure is formed because all places $p \in \mathcal{E}(Post(t,.))$ of a transition t are presented at the same abstraction level. Elements in $\mathcal{E}(Pre(.,t))$ belong to the same and/or to an lower abstraction level than elements in $\mathcal{E}(Post(t,.))$. Ordered monotonic refinement assures us that if a marking $\mathcal{A}_i(M_g), i > 0$ is not reachable at abstraction level i, M_g would not be reachable in the original Petri net N either. Therefore we do not need to explore the entire search space to get acknowledged about it.

Every abstracted net at level i includes either at least one token from the initial marking ($\mathcal{E}(\mathcal{A}_i(N)) \not\equiv \emptyset$) or at least one source transition. Every source transition represents a transition wherefrom tokens may enter the subnet. Therefore, if the marking $\mathcal{A}_i(M_g)$ is not reachable under these relaxing conditions, the marking $\mathcal{A}_i(M_g)$ would not be reachable without particular relaxing conditions either.

3.2 The Role of the Initial Marking

While building a dependency graph for abstraction, dependencies between Petri net places are detected. If it should happen that at least one place $p \in \mathcal{E}(M_0)$ is not included in the dependency graph and it does not occur in $\mathcal{E}(M_g)$ either, then there is no sequence of transitions s that $M_0 \xrightarrow{s} M_g$. This applies iff there are no sink transitions, which could consume tokens in p.

Theorem 1. *Given a Petri net N and a set of edges D_e of dependency graph $\mathcal{D}$, which was constructed using N and M_g, then if it is satisfied that $\exists p.(p \in \mathcal{E}(M_0) \wedge p \notin \mathcal{E}(M_g) \wedge p \notin D_e \wedge \neg\exists t.\mathcal{E}(Post(t,.)) \equiv \emptyset)$, then goal marking M_g is not reachable from marking M_0 of Petri net N.*

Proof. While finding dependencies between Petri net places through dependency graph construction, roughly a way for token propagation is estimated for reaching the marking M_g and places on the way are inserted to the graph. Therefore, if not all places $p \in \mathcal{E}(M_0)$ are included in the dependency graph, then there is no way to reach from the marking M_0 the marking M_g.

Anyway, some tokens in the initial marking may be not fired during reachability checking and thus they exist both in markings M_0 and M_g. In that case the missing place from a dependency graph does not indicate that the marking M_g is not reachable. Similarily, sink transitions have to be considered, since they only consume tokens and therefore are rejected, when generating a dependency graph.

This case is illustrated in Figure 6, where a dependency graph is generated for the marking $\mathcal{M}(M_g) = \{F\}$. As it can be seen in Figure 6(b) the Petri net place B, although having a token in the marking M_0, is not included in the dependency graph in Figure 6(b). The same applies for place A. Therefore the marking M_g is not reachable.

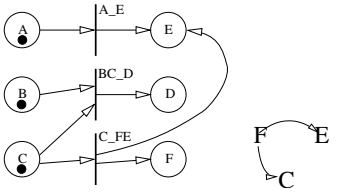


Fig. 6. A Petri net (a) with the initial marking M_0, $\mathcal{M}(M_0) = \{A, B, C\}$ and the marking M_g, $\mathcal{M}(M_g) = \{F\}$ plus the corresponding dependency graph (b)

3.3 Removing Redundant Transitions

After the places, where tokens are possibly propagated through, have been determined, we can throw away all transitions, which are connected at least to one place which is not included in the dependency graph. In that way the search space would be pruned and search made more efficient. Reachability result would not be affected by removing these transitions.

Theorem 2. *Given a Petri net N and a set of edges D_e of dependency graph $\mathcal{D}$, which was constructed using N and M_g, we can discard all transitions $t \in T$, which satisfy condition $\exists p.((p \in \mathcal{E}(Pre(.,t)) \vee p \in \mathcal{E}(Post(t,.))) \wedge p \notin D_e \wedge \mathcal{E}(Post(t,.)) \not\equiv \emptyset)$ without affecting reachability result for $M_0 \xrightarrow{s} M_g$.*

Proof. If there is a transition $t \in T$ of a Petri net N such that $\exists p.((p \in \mathcal{E}(Pre(.,t)) \vee p \in \mathcal{E}(Post(t,.))) \wedge p \notin D_e)$, then it means that the transition t was not considered during construction of dependency graph $\mathcal{D}$. Therefore the transition is not considered relevant for achieving marking M_g and can be discarded.

Transition reduction is illustrated in Figure 7, where a dependency graph is generated for the marking $M_g, \mathcal{M}(M_g) = \{F, E^2\}$. Since B and D are not present in the dependency graph, they are considered irrelevant for the current

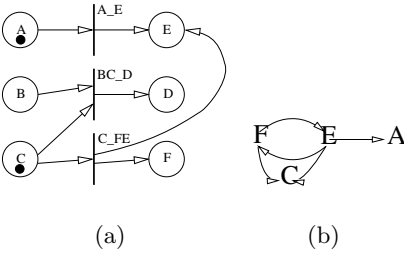


Fig. 7. A Petri net (a) including redundant transitions for $M_g, \mathcal{M}(M_g) = \{F, E^2\}$ plus the corresponding dependency graph (b)

reachability problem. Therefore all transitions t such that $Pre(B,t) \in Pre$ or $Pre(D,t) \in Pre$ or $Post(t,B) \in Post$ or $Post(t,D) \in Post$ can be removed without affecting the reachability result of the original problem. Hence the transition BC_D can be removed.

3.4 The Computational Complexity of Abstraction

According to [7] the complexity of building the dependency graph is $O(n * o * l)$, where n is the number of different places in a Petri net, o is the maximum number of transitions t such that for any place p $Pre(p,t) > 0$ or $Post(t,p) > 0$, and l is the maximum number of different places p such that $Post(t,p) > 0$ with any transition t. The complexity of building an hierarchy is also $O(n * o * l)$, since the complexity of the graph algorithms is bounded by the number of edges, which is bounded in our case by $n * o * l$.

4 Reachability Checking with Abstraction

After an abstraction hierarchy has been generated, we start hierarchical reachability checking from the highest abstraction level by mapping the original Petri net N to $\mathcal{A}_h(N)$, where h denotes the highest abstraction level. First we find a sequence s_h of transitions $t \in T_h$ so that $\mathcal{A}_h(M_0) \xrightarrow{s_h} \mathcal{A}_h(M_g)$. Then we gradually refine this sequence while moving lower in the abstraction hierarchy until we reach the lowest level of the abstraction hierarchy. If it should turn out that from a certain abstraction level there is no refinement to a lower abstraction level for a sequence, then the marking M_g is considered not reachable and search is halted.

Definition 4. *New transitions $t \in T_{new_i}$ at abstraction level i are defined as $T_{new_i} = T_i \setminus (T_i \cap T_{i+1}), i = 0 \ldots n$, with exception $T_{new_n} = T_n$, where n is the highest abstraction level, T_i is a set of transitions in a Petri net $\mathcal{A}_i(N)$ and N is the original Petri net.*

It is crucial to note that at every abstraction level i only transitions $t \in T_{new_i}$ may be used during refinement. This is the basic search space reduction mechanism, supported by abstraction, which divides the initial search space into subspaces.

A sequence of transitions, which shows the reachability of a marking M_g, found at an abstraction level higher than 0 may be viewed as a sequence including "gaps", which have to be filled at a lower level of abstraction. It has to be emphasised that at one abstraction level several solutions may be found and not all of these, if any, lead to a solution at less abstract level. Thus several abstract solutions may have to be tried before a solution for less abstract problem is found.

The ordered monotonicity determines that while extending a firing sequence s at a lower abstraction level with transitions, we can insert only *new* transitions, whereas the transitions which are already in s, after enriching their rep-

resentation with places permitted at that abstraction level (if needed), possibly determine new submarking reachability problems we have to solve in order to solve the general reachability problem. Thus in that way we reduce one single reachability problem into several reachability problems and reduce distance between submarkings. By dividing transitions between different abstraction levels the branching factor of search space is decreased.

Following the former idea we define the optimal abstraction hierarchy as an abstraction hierarchy, where at each level exactly one new transition is introduced and a Petri net at the highest abstraction level includes exactly one transition.

Definition 5. *An optimal abstraction hierarchy $\mathcal{H}_o$ of a Petri net N is an abstraction hierarchy with $n = |T|$ abstraction levels starting from level 0. Therefore, in $\mathcal{H}_o$, $|T_i \setminus (T_i \cap T_{i+1})| = 1, i = 0 \ldots n-2$ and $|T_{n-1}| = 1$.*

Theorem 3. *Given that an optimal abstraction hierarchy $\mathcal{H}_o$ is used, computational complexity of solving a reachability problem $M_0 \xrightarrow{s} M_g$ of a Petri net N with our algorithm is $O(|T| * |s|)$, where $|T|$ is the number of transitions in a net and $|s|$ is the expected length of the firing sequence s.*

Proof. We define the exponential complexity of Petri net reachability checking as $l_t^{l_s}$, where l_t is the number of transitions in a Petri net N and $l_s = |s|$ is the length of a transition firing sequence s such that $M_0 \xrightarrow{s} M_g$. Since at the base abstraction level (level 0) of $\mathcal{H}_o$ we have l_s transitions in the sequence s, there are at every abstraction level maximally l_s gaps, which have to be filled. By assuming that there are l_h abstraction levels in $\mathcal{H}_o$, the resulting complexity is $O(l_h * l_s * (l_t/l_h)^{l_s})$. Since we assumed usage of an *optimal* abstraction hierarchy ($l_t \equiv l_h$), the exponential complexity of Petri net reachability checking is reduced to $O(l_h * l_s * 1^{l_s}) = O(l_h * l_s)$, which is polynomial.

5 Experimental Results for Abstraction-Based Reachability Checking

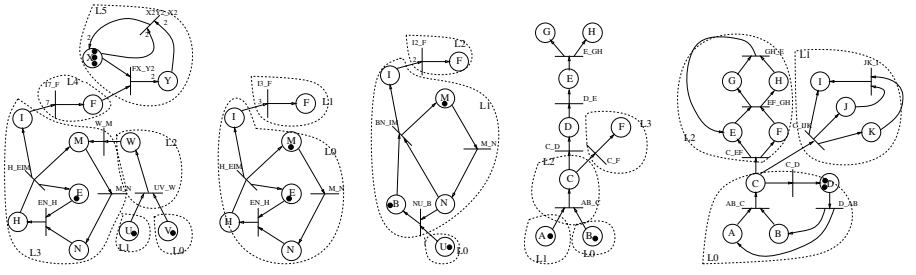
All experiments were performed with the RAPS tool[1], where we applied basically on-the-fly depth-first search over a reachability graph. The maximum search depth was 30. The results are summarised in Table 1.

Columns covered by labels "Collapsing" (state space collapsing with Karp-Miller algorithm [6]) and "No collapsing" include numbers, which represent how many Petri net transitions were fired before reachability was detected. "—" in fields of the table indicates that reachability was not detected. This illustrates that, if we apply Karp-Miller algorithm together with abstraction, we may not discover that a marking is reachable.

[1] Downloadable from `http://www.idi.ntnu.no/~peep/RAPS`.

Table 1. Reachability checking with and without abstraction

Problem	Collapsing		No collapsing		Sol length	Abs levels	$\|T\|$	$\|P\|$	$\mathcal{M}(M_g)$
	Abs	No abs	Abs	No abs					
Figure 4(a)	8	179	26	81	24	3	6	8	$\{E, M, X^2\}$
Figure 8(a)	—	181	28	69	26	6	8	11	$\{E, M, X^2\}$
Figure 8(b)	4	17	10	99	10	2	4	6	$\{E, M, F\}$
Figure 8(c)	6	8	6	8	6	3	4	6	$\{F, M\}$
Figure 8(d)	2	5	2	5	2	4	5	8	$\{F\}$
Figure 8(e)	—	22	9	22	9	3	8	11	$\{E, I^2\}$



(a) 1. (b) 2. (c) 3. (d) 4. (e) 5.

Fig. 8. Sample nets used in experiments

Columns "Problem", "Sol length" and "Abs levels" represent information respectively about net descriptions, firing sequence lengths and the number of abstraction levels per problem, if abstraction was used. "Abs" and "No abs" distinguish whether abstraction was used or not. "Collapsing" and "No collapsing" in turn determine whether state space collapsing [6] was applied or not. $|T|$ and $|P|$, as defined in Section 2, in column headers denote to the numbers of transitions and places in a sample net.

Dashed lines in figures of Figure 8 separate different abstraction levels. Thus it is easy to follow how abstraction levels were formed for different nets and reachability problems. If a fragment of a net is not encircled with a dashed line, then this part is discarded as irrelevant for a given reachability problem. The letter L followed by a number indicates there the abstraction level of a particular subnet.

Although one may argue that only toy examples were considered while perfoming experiments, these examples still illustrate advantages of abstraction-based reachability checking over ordinary reachability checking. One can see that the difference between "ordinary" and abstraction-based reachability checking may be up to 22 times in the current case. These experimental results encourage us to apply abstraction-based reachability checking for larger problems.

While experimenting with abstraction and Petri nets we experienced cases, where with abstraction, although more Petri net transitions were fired, less overall time was spent on it than with ordinary reachability checking. This "anomaly" arises from the way we probe whether a transition is enabled or not. In the case of ordinary reachability analysis one vector is subtracted from another and then it is checked whether the result is not negative.

Anyway, if we use abstraction, then transitions indicate to a specific abstraction level where they may be applied only. In that case a lot of transitions are disqualified just by comparing the integer indicating the abstraction level where the transition may be fired. And comparing two integers is computationally cheaper than comparing two state vectors. Therefore, by using abstraction, we do not search blindly anymore, but use instead abstraction as search heuristics.

6 Related Work

Abstraction of Petri nets has been explored previously by several researchers [14, 13, 15]. These approaches are based on analysing structural properties of a net. Abstraction is performed by substituting subnets with single transitions or places.

Berthelot [2] considers Petri net reduction by applying a set of transformation rules. Although this work does not consider abstraction itself, some of the transformation rules may be viewed as abstraction operators.

Several abstraction techniques have been proposed in AI planning and theorem proving disciplines. The first explicit use of abstraction in automated deduction was in the planning version of GPS [12]. Other approaches to automatic generation of abstraction spaces in the AI planning field include [1, 3, 9].

In [5] another abstraction technique, STAR, for AI planning is proposed. Unfortunately it abstracts the generated state space and not the initial problem representation. The main idea there is to collapse a state and its adjacent states into a single one thereby reducing the state space. Although this methodolody may turn out to be useful for applications, where a Petri net state space is first generated and then analysed, we are interested in on-the-fly reachability checking.

7 Conclusions

In this paper we presented an algorithm for automatically abstracting Petri nets. While other approaches of abstracting Petri nets are based merely on the structural properties of nets, we considered also the initial and the final marking, whose reachability has to be checked. Our algorithm first generates an abstraction hierarchy, which divides an original Petri net into several abstracted representations. These abstracted nets are ordered into an abstraction hierarchy by their size.

Abstraction hierarchies are generated by observing connections between transitions and places of nets. Given a net and the final marking, first a dependency

graph is generated, which determines transitions, possibly to be fired in order to reach the final marking. Then this graph is transformed into abstraction hierarchies.

The greatest benefits of our abstraction and reachability checking algorithms are achieved on tree-like Petri net structures like the one depictured in Figure 5. If a whole Petri net represents a tree, optimal abstraction hierarchies are constructed. By using optimal abstraction hierarchies the computational complexity of reachability checking is reduced to polynomial.

It turns out that the dependency graph, which is a byproduct of the abstraction algorithm, can be used also to determine these transitions in a net, which are not relevant for solving a given reachability problem. In some cases the dependency graph also helps to determine the lack of a solution even before starting with reachability checking.

Additionally we sketched an algorithm for using generated abstraction hierarchies for reachability checking. Reachability checking starts from the highest abstraction level. Then a firing sequence of transitions reaching a particular marking is gradually extended until the lowest abstraction level is reached.

Finally experimental results were presented, which show that while using our abstraction methodology roughly up to 20 times less Petri net transitions are fired during reachability checking. These results motivate us to proceed with research on abstraction-based reachability checking.

Acknowledgements

This work was partially supported by the Norwegian Research Foundation in the framework of Information and Communication Technology (IKT-2010) program—the ADIS project. The author would like to thank the anonymous referees for their comments.

References

1. J. S. Anderson and A. M. Farley. Plan abstraction based on operator generalization. In *Proceedings of the Seventh National Conference on Artificial Intelligence, Saint Paul, MN, 1988*, pages 100–104, 1988.
2. G. Berthelot. Transformations and decompositions of nets. In *Petri Nets: Central Models and Their Properties, Advances in Petri Nets '86, West Germany, Bad Honnef, September 1986*, volume 254 of *Lecture Notes in Computer Science*, pages 359–376. Springer-Verlag, 1987.
3. J. Christensen. *Automatic Abstraction in Planning.* PhD thesis, Department of Computer Science, Stanford University, 1991.
4. J. Esparza and M. Nielsen. Decidability issues for Petri nets—a survey. *Journal of Information Processing and Cybernetics*, 30:143–160, 1995.
5. R. C. Holte, T. Mkadmi, R. M. Zimmer, and A. J. MacDonald. Speeding up problem solving by abstraction: A graph oriented approach. *Artificial Intelligence*, 85:321–361, 1996.

6. R. M. Karp and R. E. Miller. Parallel program schemata. *Journal of Computer and Systems Sciences*, 3(2):147–195, 1969.
7. C. A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68:243–302, 1994.
8. R. E. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33:65–88, 1987.
9. A. Y. Levy. Creating abstractions using relevance reasoning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, pages 588–594, 1994.
10. R. J. Lipton. The reachability problem requires exponential space. Research Report 62, Department of Computer Science, Yale University, 1976.
11. T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of IEEE*, 77(4):541–580, 1989.
12. A. Newell and H. A. Simon. *Human Problem Solving*. Prentice-Hall, 1972.
13. J.-S. Song, S. Satoh, and C. V. Ramamoorthy. The abstraction of petri net. In *Proceedings of TENCON'87, Seoul, Korea, August 25–28, 1987*, pages 467–471. IEEE Press, 1987.
14. I. Suzuki and T. Murata. A method for stepwise refinement and abstraction of petri nets. *Journal of Computer and System Sciences*, 27:51–76, 1983.
15. R. Vallette. Analysis of petri nets by stepwise refinement. *Journal of Computer and System Sciences*, 18(1):35–46, 1979.

Detecting and Breaking Symmetries by Reasoning on Problem Specifications

Toni Mancini and Marco Cadoli

Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza"
{tmancini, cadoli}@dis.uniroma1.it

Abstract. In this paper we address the problem of detecting and breaking symmetries in combinatorial problems, following the approach of imposing additional symmetry-breaking constraints. Differently from other works in the literature, we attack the problem at the *specification* level. In fact, many symmetries depend on the structure of the problem, and not on the particular input instance. Hence, they can be easily detected by reasoning on the specification, and appropriate symmetry-breaking formulae generated. We give formal definitions of symmetries and symmetry-breaking formulae on specifications written in existential second-order logic, clarifying the new definitions on some specifications: Graph 3-coloring, Social golfer, and Protein folding problems. Finally, we show experimentally that, applying this technique, even if in a naive way, to specifications written in state-of-the-art languages, e.g., OPL, may greatly improve search efficiency.

1 Introduction

The presence of symmetries in constraint satisfaction problems (CSPs) has been widely recognized to be one of the major obstacles for their efficient resolution. Much work has been already done, and a wide literature is nowadays available on how symmetries can be exploited, with the aim of greatly reducing the size of the search space. There are four main approaches followed by the research community to deal with symmetries:

1. Imposing additional constraints on the problem model, which are satisfied only for one of the symmetrical points in the search space, cf., e.g., [21, 7, 9];
2. Introducing additional constraints during the search process, to avoid the traversal of symmetrical points, cf., e.g., [3, 10];
3. Defining a search strategy able to break symmetries as soon as possible (e.g., by first selecting variables involved in the greatest number of local symmetries), cf., e.g., [18];
4. Isolating subclasses of CSPs for which particular search strategies can be used in order to efficiently break their symmetries (cf., e.g., tractability of symmetry breaking for CSPs with various form of interchangeability [25]).

However, all these approaches make the assumption that symmetries of the constraint problem at hand are known. Hence, the problem of the *automatic detection* of symmetries arises. Currently, symmetry detection is either performed

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 165–181, 2005.

by hand (it is the modeller that states them, by analyzing the problem), or recognized by reducing the CSP obtained *after* instantiation to an instance of the graph automorphism problem (for which there are no polynomial time algorithms, even if there is evidence that it is not NP-complete [13]).

On the other hand, many of the available systems and languages for the solution of constraint problems (e.g., AMPL [11], OPL [24], XPRESS^{MP}[1], DLV [15], SMODELS [19], and NP-SPEC [6]) clearly separate the *specification* of a problem from its *instances*. Furthermore, symmetries often arise from the problem structure, and not from the particular instance considered. Hence, they often clearly emerge at the compact, symbolic level of the specification. Nonetheless, many of the existing approaches to automatic symmetry detection (cf., e.g., the package *Nauty* [16]) try to infer all symmetries of a constraint problem after instantiation, where many structural aspects have been irremediably hidden.

In our opinion, reasoning at the logical level of the problem specification may be much effective in order to detect those structural properties that are suitable for optimization and reformulation, as many symmetries are: problem specifications are usually much more compact, readable, and high-level modelled, hence the recognition of, e.g., structural symmetries naturally fits at this stage. Moreover, convenient symmetry-breaking formulae (cf. approach 1 in the list above) can be added to the specification itself in order to exploit them. Finally, since specifications are logical formulae, computer tools can be used to automatically or semiautomatically detect and break symmetries [5].

Such reasoning tasks have, in principle, at least two applications: *(i)* Discover new properties of a specification, and *(ii)* Validate a specification confirming the existence of some properties. In this paper, we mainly focus on the latter, giving a formal characterization of symmetries and symmetry-breaking formulae for a specification. This is a mandatory first step also to solve *(i)* (an heuristic, and incomplete, approach for detecting some symmetries on specifications is discussed in [26]).

Of course, detecting and breaking symmetries at the specification level does not rule out the possibility to compositionally use symmetry-breaking techniques at the instance level (e.g., [7, 9]), in order to deal with additional symmetries that arise from the problem instance. As an example, since some systems generate a SAT instance, e.g., [6], or an instance of integer linear programming, e.g., [24], it is possible to do symmetry breaking on such instances, using existing techniques.

2 Existential Second-Order Logic as a Modelling Language

When dealing with problem specifications, the first choice to be made is that of the modelling language to be used. Current systems and languages for declarative constraint modelling, as those listed in Section 1, have their own syntax for describing problem specifications: AMPL, OPL, and XPRESS^{MP} allow the

[1] cf. `http://www.dashoptimization.com`.

representation of constraints by using algebraic expressions, while others, e.g., DLV, SMODELS, and NP-SPEC are rule-based languages, more specifically extensions of datalog. Anyway, from an abstract point of view, all such languages are extensions of *existential second-order logic* (ESO) over finite databases, where the existential second-order quantifiers and the first-order formula represent, respectively, the *guess* and *check* phases of the constraint modelling paradigm. In particular, even if all such languages have a richer syntax and more complex constructs, in all of them it is possible to embed ESO queries, and the other way around is also possible, as long as only finite domains are considered. Hence, as we show in the remainder of this section, *ESO can be considered as the formal logical basis for virtually all available languages for constraint modelling*, being able to represent all search problems in the complexity class NP [20]. Moreover, since checking and breaking symmetries on ESO specifications reduces to check semantic properties of logical formulae, it is possible to use known results and techniques in order to automate such tasks.

Formally, an ESO specification describing a search problem π is a formula $\psi_\pi \doteq \exists \boldsymbol{\mathcal{S}}\ \phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{R}})$, where $\boldsymbol{\mathcal{R}} = \{R_1, \ldots, R_k\}$ is the input relational schema (i.e., a fixed set of relations of given arities denoting the schema for all input instances for π), and ϕ is a closed first-order formula on the relational vocabulary $\boldsymbol{\mathcal{S}} \cup \boldsymbol{\mathcal{R}} \cup \{=\}$ ("=" is always interpreted as identity). An instance $\boldsymbol{\mathcal{I}}$ of the problem is given, as it happens in current systems, as a relational database over the schema $\boldsymbol{\mathcal{R}}$, i.e., as an extension for all relations in $\boldsymbol{\mathcal{R}}$. Predicates (of given arities) in the set $\boldsymbol{\mathcal{S}} = \{S_1, \ldots, S_n\}$ are called *guessed*, and their possible extensions (with tuples on the domain given by constants occurring in $\boldsymbol{\mathcal{I}}$ plus those occurring in ϕ, i.e., the so called Herbrand universe) encode points in the search space for problem π on instance $\boldsymbol{\mathcal{I}}$. Formula ψ_π correctly encodes problem π if, for every input instance $\boldsymbol{\mathcal{I}}$, a bijective mapping exists between solutions to π and extensions of predicates in $\boldsymbol{\mathcal{S}}$ which verify $\phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{I}})$. More formally, the following must hold:

$$\text{For each instance } \boldsymbol{\mathcal{I}}\text{:} \quad \boldsymbol{\Sigma} \text{ is a solution to } \pi(\boldsymbol{\mathcal{I}}) \iff \{\boldsymbol{\Sigma}, \boldsymbol{\mathcal{I}}\} \models \phi.$$

It is worthwhile to note that, when a specification is instantiated, a constraint satisfaction problem (CSP) is obtained.

In order to facilitate the writing of specifications, several built-in constructs are provided by current languages, in particular those for typed relations, functions (cf., e.g., arrays), bounded integers and arithmetics over them. Hence, to ease expressions, and to make specifications more compact and closer to their counterparts in state-of-the-art languages, in this paper we consider an enriched ESO. In particular, we assume that:

1. Guessed predicates may be typed: we write $\exists S \in type_S^1 \times \cdots \times type_S^k$, where each $type_S^i$ is a monadic relation in $\boldsymbol{\mathcal{R}}$ that represents the domain of the i-th argument of S. (For simplicity of notation, given a relation S of arity k, we denote with $type(S)$ the domain of tuples that belong to S, i.e., the set $type_S^1 \times \cdots \times type_S^k$.)
2. Guessed predicates that encode functions can be natively expressed in the language (we write $\exists S \in type_S^1 \times \cdots \times type_S^j \rightarrow type_S^{j+1} \times \cdots \times type_S^k$ for some $j \in [1, k-1]$. Total functions will be denoted by "(total)").

3. Bounded integers and arithmetics over them are available.

We note that such additions do not change the expressive power of the language. Types and (total) functions can be simulated in ESO by means of monadic predicates in $\boldsymbol{\mathcal{R}}$ and first-order constraints, respectively. The same holds for bounded integers and arithmetics (that can be pre-interpreted).

Formally, we denote the set of monadic relations that encode types as $\boldsymbol{\mathcal{T}}$ (with $\boldsymbol{\mathcal{T}} \subseteq \boldsymbol{\mathcal{R}}$). Hence, a specification in the enriched language is of the kind:

$$\exists S_1 \in type(S_1), \ldots \exists S_n \in type(S_n) \quad \phi(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{R}}) \tag{1}$$

where $type(S_i) = type^1_{S_i} \times \cdots \times type^{ar(S_i)}_{S_i}$, with all $type^j_{S_i} \in \boldsymbol{\mathcal{T}}$.

Since $\boldsymbol{\mathcal{T}} \subseteq \boldsymbol{\mathcal{R}}$, we normally omit $\boldsymbol{\mathcal{T}}$ as argument of ϕ, even if, in some cases, in order to emphasize the occurrence of types relations in some formulae, we state it explicitly.

Example 1 (Graph k-coloring). Given an undirected graph and a set of k colors as input, this problem amounts to decide whether it is possible to give each of its nodes one out of the colors, in such a way that adjacent nodes (not including self-loops) are never colored the same way. The problem is well-known to be NP-complete for $k \geq 3$, and can be specified in ESO by, e.g., the following formula over relations in $\boldsymbol{\mathcal{R}} = \{node(\cdot), edge(\cdot,\cdot), color(\cdot)\}$, listing the graph nodes, edges and the available colors, respectively. The set of types $\boldsymbol{\mathcal{T}}$ is given by $\{node, color\}$. In particular, relation $color(\cdot)$ will have exactly k tuples. We also assume that $node(\cdot)$ and $color(\cdot)$ have no tuples in common.

$$\exists Col \ \in \ node \rightarrow color \text{ (total)} \tag{2}$$

$$\forall X, Y, C, C' \quad edge(X,Y) \wedge X \neq Y \wedge Col(X,C) \wedge Col(Y,C') \ \rightarrow \ C \neq C'. \tag{3}$$

Part (2) of the above specification defines *Col* as a total function assigning a color to each node, while (3) is the good coloring constraint. It is worth noting that the specification above is very close to that written in available languages, e.g., the following one in OPL (initializations are omitted):

```
range node 1..n_nodes;  range color 1..n_colors;
var color Col[node];
solve { forall (e in edges: e.start<>e.end) Col[e.start]<>Col[e.end]; };};
```

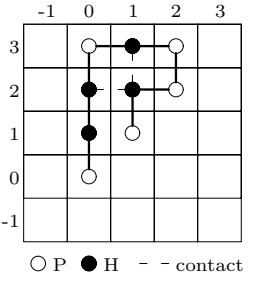
Another assumption that we make in this paper is that the set of guessed predicates $\boldsymbol{\mathcal{S}}$ is partitioned in two parts: *output* and *auxiliary* guessed predicates, denoted, respectively, as $\boldsymbol{\mathcal{O}}$ and $\boldsymbol{\mathcal{A}}$ (with $\boldsymbol{\mathcal{A}}$ possibly empty). Output guessed predicates conceptually denote the search space, while auxiliary predicates are used internally to maintain and/or compute additional information needed to express and evaluate the constraints. This is a very common necessity in declarative languages, as forthcoming Example 2 shows.

When such a partition is made, a solution is completely characterized by the extensions of predicates in $\boldsymbol{\mathcal{O}}$ and *not* by those of predicates in $\boldsymbol{\mathcal{A}}$. Hence, the general form of a problem specification in ESO is as follows:

$$\exists \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}} \quad \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \tag{4}$$

where predicates in $\boldsymbol{\mathcal{O}}$ and $\boldsymbol{\mathcal{A}}$ may have an associated type, that can (in general) be represented with a first-order formula over $\boldsymbol{\mathcal{T}}$.

Example 2 (HP 2D-Protein folding [14]). This specification models a simplified version of one of the most important problems in computational biology. It consists in finding the spatial conformation of a protein (i.e., a sequence of amino-acids) with minimal energy. The simplifications are twofold: the amino-acids alphabet is reduced to just H (*hydrophobic*) and P (*polar*), and the protein is forced to fold in a 2D discrete space. However, the simplified problem is known to be NP-complete [8]. Given the sequence of amino-acids of the protein, i.e., a string over {H,P} of length n, the problem aims to find a connected shape for it on a 2D grid (with coordinates in $[-(n-1), (n-1)]$, starting at $(0,0)$), non-crossing, and such that the number of "contacts", i.e., the number of non-sequential pairs of Hs for which the Euclidean distance of the positions is 1 is maximized (the overall energy is the opposite of the number of contacts). The figure below shows a possible conformation of the protein "PHHPHPPHP", with overall energy -2.



Different alternatives for the search space obviously exist: as an example, we can guess the position on the grid of each amino-acid, and then force the shape to be connected, non-crossing, and with minimal energy. However, a preferred approach that reduces the size of the search space (4^n points versus $(2n)^{2n}$) is to guess the shape of the protein as a connected path starting at $(0,0)$, by guessing, for each index $i \in [1, n-1]$, the direction that the $(i+1)$-th amino-acid assumes wrt the i-th one (directions can only be North, South, East, West).

The extension of *Move* for the shape in the figure is: $\{\langle 1, N\rangle, \langle 2, N\rangle, \langle 3, N\rangle, \langle 4, E\rangle, \ldots\}$. However, choosing the latter model is not completely satisfactory: to express the non-crossing constraint, and to compute the number of contacts, absolute coordinates of each amino-acid must be computed and maintained. An ESO specification for this problem, where, for simplicity, we assume to deal with its decisional version, and to have (pre-interpreted) bounded integers and arithmetics in the range $[-(n-1), n-1]$, is as follows ($\boldsymbol{\mathcal{R}} = \{index(\cdot), elem(\cdot,\cdot)\}$, with $elem(i,a)$ stating that the i-th element of the protein is $a \in \{H, P\}$):

$$\exists Move \ \in \ index \rightarrow \{N, S, E, W\} \text{ (total)}, \tag{5}$$

$$\exists X, Y \ \in \ index \rightarrow [-n+1, n-1] \text{ (total)} \tag{6}$$

$$X(0,0) \ \wedge \ Y(0,0) \ \wedge \tag{7}$$

$$\begin{aligned} &\forall I, I' \ index(I) \wedge index(I') \wedge I' = I - 1 \ \rightarrow \\ &\quad \forall D, X, Y, X', Y' \ Move(I', D) \wedge X(I, X) \wedge X(I', X') \wedge Y(I, Y) \wedge Y(I', Y') \rightarrow \\ &\qquad D = N \ \rightarrow X = X' \wedge Y = Y' + 1 \wedge \\ &\qquad D = S \ \rightarrow X = X' \wedge Y = Y' - 1 \wedge \\ &\qquad D = E \ \rightarrow X = X' + 1 \wedge Y = Y' \wedge \\ &\qquad D = W \rightarrow X = X' - 1 \wedge Y = Y' \wedge \end{aligned} \tag{8}$$

$$\begin{aligned} &\forall I, I', X, X', Y, Y' \\ &\quad I \neq I' \wedge X(X, I) \wedge X(X', I') \wedge Y(Y, I) \wedge Y(Y', I') \ \rightarrow \ X \neq X' \vee Y \neq Y' \wedge \end{aligned} \tag{9}$$

$$\left| \left\{ \begin{array}{c} \langle I, I' \rangle \mid index(I) \wedge index(I') \wedge (I+1 < I') \wedge elem(I, H) \wedge elem(I', H) \wedge \\ \forall X, X', Y, Y' \ X(X, I) \wedge X(X', I') \ \wedge \\ [3pt] Y(Y, I) \wedge Y(Y', I') \ \wedge \ |X - X'| + |Y - Y'| = 1 \end{array} \right\} \right| \geq k \tag{10}$$

Constraints (5) and (6) declare guessed predicates *Move*, X and Y as total functions assigning, respectively, a value in $\{N, S, E, W\}$, and a value in $[-(n-1), n-1]$ to each amino-acid.[2] Furthermore, (7) forces the first amino-acid to be placed in $(0, 0)$, while (8) defines the absolute position of each amino-acid starting from that of the previous one and the move. Finally, (9) is the non-crossing constraint, and (10) forces the number of contacts to be at least k (integer k is assumed to be fixed). The above specification is very similar to that given in available languages, e.g., OPL (cf. [4]).

From the problem description, *Move* is the output guessed predicate, while X and Y are auxiliary: a solution is completely characterized by the sole extension of *Move*. However, it is a matter of choice and responsibility of the modeler to state which guessed predicates are output and which others are auxiliary, and, of course, it is always possible to consider all guessed predicates as output ones (hence, $\boldsymbol{\mathcal{A}}$ can always be empty). Indeed, in the following we show that this "conceptual" partition plays an important role in detecting structural properties, e.g., symmetries, that may be exploited to improve efficiency. We also observe that, in this example, X and Y are functionally dependent on *Move* (cf. [4]).

3 Symmetries on Problem Specifications

In this section we define the concepts of *transformation* and *symmetry* on a specification, and investigate interesting specializations of them. In Section 2, we presented some syntactic sugar that can be added to ESO in order to have more compact and readable specifications. However, we also noticed that such constructs can always be regarded as additional constraints. Hence, for what concerns the reasoning tasks that we describe from this section on, we consider the basic ESO framework. Hence, all such additional constraint will be considered as integral part of the ϕ part of a specification having the general form (4).

Definition 1 (Transformation). *Given a specification* $\exists \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}})$*, a* transformation *for* $\boldsymbol{\mathcal{O}}$ *and* $\boldsymbol{\mathcal{A}}$ *is a family of functions, one for each possible finite Herbrand domain* $\mathcal{H}$*, of the kind* $\tau_{\mathcal{H}}$*:*$\{ext_{\mathcal{H}}(\boldsymbol{\mathcal{O}},\boldsymbol{\mathcal{A}})\} \rightarrow \{ext_{\mathcal{H}}(\boldsymbol{\mathcal{O}},\boldsymbol{\mathcal{A}})\}$*, where* $\{ext_{\mathcal{H}}(\boldsymbol{\mathcal{O}},\boldsymbol{\mathcal{A}})\}$ *is the set of all possible extensions of predicates in* $\boldsymbol{\mathcal{O}}$ *and* $\boldsymbol{\mathcal{A}}$ *with elements in* $\mathcal{H}$*.*

Intuitively, a transformation is a mapping from and to all points in the search space defined by all the guessed predicates in the specification, for any $\mathcal{H}$. For the sake of simplicity, and with a little abuse of notation, in what follows we denote a transformation as a single function τ: $\{ext(\boldsymbol{\mathcal{O}},\boldsymbol{\mathcal{A}})\} \rightarrow \{ext(\boldsymbol{\mathcal{O}},\boldsymbol{\mathcal{A}})\}$, obtained by collapsing all the $\tau_{\mathcal{H}}$, which is defined on all finite Herbrand domains $\mathcal{H}$.

By focusing only on the set $\boldsymbol{\mathcal{O}}$, the following definition holds:

[2] Actually, *Move* should be not defined for the last amino-acid. However, the proposed simpler specification remains correct, with the last move having no meaning.

Definition 2 (Symmetry). *Given a problem specification* $\exists \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}})$ *as above, a* symmetry *is an* invertible *transformation for* $\boldsymbol{\mathcal{O}}$*, i.e., an invertible function* $\tau : \{ext(\boldsymbol{\mathcal{O}})\} \rightarrow \{ext(\boldsymbol{\mathcal{O}})\}$ *such that, for every input instance* $\boldsymbol{\mathcal{I}}$*, and every extension* $\boldsymbol{\Omega}$ *for relations in* $\boldsymbol{\mathcal{O}}$*, the following holds:*

$$\boldsymbol{\Omega}, \boldsymbol{\mathcal{I}} \models \exists \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \iff \tau(\boldsymbol{\Omega}), \boldsymbol{\mathcal{I}} \models \exists \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}). \tag{11}$$

The distinction between output and auxiliary guessed predicates here becomes more clear: a symmetry is a transformation of the sole output predicates such that, if an extension of $\boldsymbol{\mathcal{O}}$ may lead to a solution (with appropriate extensions for auxiliary predicates $\boldsymbol{\mathcal{A}}$), then its transformation must also lead to a solution (even if the corresponding extensions for predicates in $\boldsymbol{\mathcal{A}}$ change) and vice versa. As an example, in the Protein folding problem, given a solution, i.e., a move in $\{N, S, E, W\}$ for each element of the sequence such that all constraints are satisfied, we can uniformly change N with S, and/or E with W and obtain another solution, even if the corresponding extensions for X and Y change.

Definition 1 is about transformations in general, but does not limit in any way the kind of functions τ. By imposing some restrictions on τ, interesting specializations arise. In this paper, we consider functions τ that focus on a single output guessed predicate, being the identity function on the others (we call them *single-predicate transformations*). They are of special interest, because of the usual structure of constraint problems, in which transformations we are interested in (i.e., candidate symmetries) often are internal to a guessed predicate.

Definition 3 (Single-predicate transformation). *A transformation is* single-predicate *if there exist* $O \in \boldsymbol{\mathcal{O}}$ *and a function* $\tau_O : \{ext(O)\} \rightarrow \{ext(O)\}$ *such that, for all extensions* $\Omega_1, \ldots, \Omega, \ldots \Omega_n$ *for* $O_1, \ldots, O, \ldots, O_n$ *(for any finite* $\mathcal{H}$*), we have that* $\tau(\Omega_1, \ldots, \Omega, \ldots \Omega_n) = \langle \Omega_1, \ldots, \tau_O(\Omega), \ldots, \Omega_n \rangle$.

A single-predicate transformation over $O \in \boldsymbol{\mathcal{O}}$ is completely characterized by giving τ_O. Further specializations of single-predicate transformations are *column* (definition omitted) and *uniform column transformations*.

Definition 4 (Uniform column transformation (UCT)). *A single-predicate transformation* τ_O *is a UCT if there exists a partition of the indexes of arguments of* O *in two (disjoint) sets,* D *and* C*, such that, for each extension* Ω *of* O*, we have that* $\tau_O(\Omega) = \Omega'$*, where:*

$$\forall \boldsymbol{\delta} \quad \boldsymbol{\delta} \in \Omega \quad \leftrightarrow \quad \langle \boldsymbol{\delta}[D], \sigma(\boldsymbol{\delta}[C]) \rangle \in \Omega'$$

where $\sigma : type(\underset{C}{\pi}(O)) \rightarrow type(\underset{C}{\pi}(O))$ *is a total invertible function on (i.e., a* permutation *of) the domain values of arguments of* O *in set* C.

A UCT that is a symmetry is called *uniform column symmetry* (UCS). Intuitively, UCTs and UCSs change only the C components of tuples in an extension of O, leaving the others (i.e., D) unchanged. Hence, they are completely described by a permutation σ from and to the type of the C columns of O. It is worth noting that σ is *uniform*, i.e., its behavior on a tuple $\boldsymbol{\delta} \in O$ depends only on $\boldsymbol{\delta}[C]$, and not on $\boldsymbol{\delta}[D]$. A (non-uniform) column transformation/symmetry, instead, is described by a function which behavior on tuple $\boldsymbol{\delta}$ depends also on $\boldsymbol{\delta}[D]$.

Example 3 (Graph k-coloring (Example 1 continued)). We have that $\boldsymbol{\mathcal{O}} = \{Col\}$, and $\boldsymbol{\mathcal{A}} = \emptyset$. By focusing on Col, with $D = \{1\}$ and $C = \{2\}$, all permutations $\sigma : color \rightarrow color$ (where $color$ is $type(\underset{C}{\pi}(Col))$) are UCSs.

These symmetries are uniform because their σs map a given color (e.g., red) always to the same color, independently on the nodes (i.e., values in column 1).

If, after instantiation, we define the corresponding CSP with one variable for each node, with domain $[1, k]$, symmetries defined above become *uniform value symmetries* (in the sense of [17]).

Example 4 (HP 2D-Protein folding (Example 2 continued)). Let us consider the UCTs that focus on $Move$, with $D = \{1\}$ and $C = \{2\}$, i.e., permutations σ of $\{N, S, E, W\}$. As an example, the following ones are UCSs:

$\sigma(N) = N,\ \sigma(S) = S,\ \sigma(E) = W,\ \sigma(W) = E$ (flip horizontally)
$\sigma(N) = S,\ \sigma(S) = N,\ \sigma(E) = E,\ \sigma(W) = W$ (flip vertically)
$\sigma(N) = S,\ \sigma(S) = N,\ \sigma(E) = W,\ \sigma(W) = E$ (flip horizontally & vertically)
$\sigma(N) = E,\ \sigma(S) = W,\ \sigma(E) = S,\ \sigma(W) = N$ (rotation 90° clockwise)

while others are not, e.g., σ such that:

$$\sigma(N) = N, \quad \sigma(S) = E, \quad \sigma(E) = W, \quad \sigma(W) = S.$$

It is worth noting that, if we consider also X and Y as output guessed predicates, the above transformations are no longer UCSs, moving to the more general class of *multiple-predicate symmetries* (definition omitted). In fact, when permuting directions in $Move$, extensions for X and Y must change accordingly.

4 Symmetry Checking

In the previous section, we considered transformations and symmetries as functions from and to extensions of predicates in $\boldsymbol{\mathcal{O}}$. Nonetheless, in order to practically deal with transformations and symmetries, we are interested into finite representations of such functions. To this end, in what follows we assume that τ is finitely representable, e.g., in first-order logic, and, with a little abuse of notation, we will denote with $\tau(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{O}}_\tau)$ a logical representation of it.[3] Such a representation will contain also occurrences of types in $\boldsymbol{\mathcal{T}}$. However, for simplicity, we do not explicitly write such types as arguments of τ.

Theorem 1. *Let $\psi \doteq \exists \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}})$ be a specification, and τ an invertible transformation for $\boldsymbol{\mathcal{O}}$. τ is a symmetry for ψ if and only if the following formula is valid:*

$$\tau(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{O}}_\tau) \quad \rightarrow \quad [\exists \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \ \leftrightarrow\ \exists \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}_\tau, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}})]\,. \tag{12}$$

It is worth noting that the above formula is second-order, even if τ is first-order. This is because of the presence of auxiliary guessed predicates $\boldsymbol{\mathcal{A}}$, which

[3] Given extensions $\boldsymbol{\Omega}$ and $\boldsymbol{\Omega}_\tau$ for $\boldsymbol{\mathcal{O}}$ and $\boldsymbol{\mathcal{O}}_\tau$ respectively, $\tau(\boldsymbol{\Omega}, \boldsymbol{\Omega}_\tau)$ is `true` iff $\boldsymbol{\Omega}_\tau$ is the output of function τ when applied to $\boldsymbol{\Omega}$.

extensions may not coincide when applying the transformation (cf., e.g., Example 4). However, in the important case of $\boldsymbol{\mathcal{A}} = \emptyset$, the above formula reduces to a first-order one.

The above theorem naturally specializes in case of single-predicate symmetries and UCSs. In the latter case, the following holds:

Theorem 2. *Let $\psi \doteq \exists \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}\, \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}})$ be a specification, and let τ_O be a UCT on $O \in \boldsymbol{\mathcal{O}}$ with σ the relative permutation of the domain values of arguments of O in set C. τ_O is a symmetry for ψ if and only if the following formula is valid:*

$$\tau(O, O_\tau) \;\rightarrow\; \exists \boldsymbol{\mathcal{A}}\; \phi(O_1,\ldots,O,\ldots,O_n,\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{R}}) \leftrightarrow \exists \boldsymbol{\mathcal{A}}\; \phi(O_1,\ldots,O_\tau,\ldots,O_n,\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{R}}) \qquad (13)$$

with τ being: $\forall \boldsymbol{X_D}, \boldsymbol{X_C}, \boldsymbol{X_C^\sigma} \quad O(\boldsymbol{X_D}, \boldsymbol{X_C}) \wedge \sigma(\boldsymbol{X_C}, \boldsymbol{X_C^\sigma}) \;\leftrightarrow\; O_\tau(\boldsymbol{X_D}, \boldsymbol{X_C^\sigma})$, and σ a finite representation of the permutation over type$\left(\underset{C}{\pi}(O)\right)$.

However, the problem of checking symmetries is undecidable. To show this, we focus on the most restricted case of first-order definable UCTs, when $\boldsymbol{\mathcal{A}} = \emptyset$.

Theorem 3. *Checking whether a first-order definable UCT τ_O is a symmetry is undecidable, even if $\boldsymbol{\mathcal{A}} = \emptyset$.*

Of course, decidable subcases for this problem may exist, and can be possibly derived by decidability results already known in first-order and second-order logic (cf., e.g., [2]). Additionally, decidable heuristic approaches, similar to those already presented in [25, 26] can be used. However, these issues are left for future research.

Often, constraint problems exhibit *many* symmetries. In order to make the relevant checks, the procedure suggested above by Theorems 1 and 2 needs to be invoked for all of them. However, when a set of symmetries can be finitely characterized, Theorems 1 and 2 can be restated with τ being the finite representation of the *whole* set of symmetries. In the particular case of UCSs, Theorem 2 can be restated with σ being the finite representation of the whole set of permutations over type$\left(\underset{C}{\pi}(O)\right)$ that are symmetries. In these cases, $\tau(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{O}}_\tau)$ holds iff $\boldsymbol{\mathcal{O}}_\tau$ is the result of applying *any* symmetry in the set to $\boldsymbol{\mathcal{O}}$ (hence, it models a relation, and not a single function any more). The same holds for σ in case of UCSs.

Example 5 (Social golfer (`www.csplib.org`, prob. 10)). Given a set of players, a set of groups, and a set of weeks, encoded in relations $\boldsymbol{\mathcal{R}} = \{player(\cdot), group(\cdot), week(\cdot)\}$ respectively, this problem amounts to decide whether there is a way to arrange a scheduling for all weeks in *week*, such that *(i)* For every week, players are divided into equally sized groups; *(ii)* Two different players don't play in the same group more than once. A specification for this problem (assuming $|player|/|group|$, i.e., the group size, integral) is the following ($Play(P, W, G)$ states that player P plays in group G on week W):

$$\exists Play \ \in \ player \times week \rightarrow group \ \ \text{(total)} \tag{14}$$

$$\forall P,P',W,W',G,G' \quad (P \neq P' \wedge W \neq W' \wedge PLAY(P,W,G) \wedge PLAY(P',W,G)) \rightarrow \neg\,[Play(P,W',G') \wedge Play(P',W',G')] \ \wedge \tag{15}$$

$$\forall G,G',W,W' \ group(G) \wedge group(G') \wedge week(W) \wedge week(W') \ \rightarrow \ |\{P : Play(P,W,G)\}| = |\{P : Play(P,W',G')\}|. \tag{16}$$

Relation *Play* is declared as a total function assigning a group to each player on each week (14); moreover, (15) is the *meet only once* constraint, while (16) forces groups to be equally sized. The last constraint can be written in ESO using standard techniques, essentially by means of an auxiliary guessed predicate *Aux* –hence $\boldsymbol{\mathcal{A}} = \{Aux\} \neq \emptyset$– forced to encode a set of bijective functions, one between tuples of any pair of sets defined in the specification.

The following sets of UCTs that focus on *Play* are all UCSs:

1. With $D = \{1,2\}$, $C = \{3\}$, all permutations $\sigma : group \rightarrow group$ of groups;
2. With $D = \{1,3\}$, $C = \{2\}$, all permutations $\sigma : week \rightarrow week$ of weeks;
3. With $D = \{2,3\}$, $C = \{1\}$, all permutations $\sigma : player \rightarrow player$ of players.

Let us consider the set of UCSs described in point 1. A finite representation for them exists, in the form of $\tau_G(Play, Play_{\tau_G})$, defined as:

$$\forall P,W,G,G^{\sigma} \ Play_{\tau_G}(P,W,G^{\sigma}) \ \leftrightarrow \ Play(P,W,G) \wedge perm(\sigma, group) \wedge \sigma(G,G^{\sigma}),$$

with $perm(\sigma, group)$ being a first-order formula stating that σ is a permutation of domain values in *group*, i.e., $type(\underset{C}{\pi}(O)))$. A formulation for *perm* is as follows:

$$perm(\sigma,R) \ \doteq \ \forall \boldsymbol{X}, \boldsymbol{X^{\sigma}} \ \sigma(\boldsymbol{X},\boldsymbol{X^{\sigma}}) \ \rightarrow R(\boldsymbol{X}) \ \wedge \ R(\boldsymbol{X^{\sigma}}) \ \wedge \tag{17}$$

$$\forall \boldsymbol{X} \ R(\boldsymbol{X}) \ \rightarrow \ \exists \boldsymbol{X^{\sigma}} \ \sigma(\boldsymbol{X},\boldsymbol{X^{\sigma}}) \ \wedge \tag{18}$$

$$\forall \boldsymbol{X}, \boldsymbol{X^{\sigma}}, \boldsymbol{X'^{\sigma}} \ \sigma(\boldsymbol{X},\boldsymbol{X^{\sigma}}) \wedge \sigma(\boldsymbol{X},\boldsymbol{X'^{\sigma}}) \rightarrow \boldsymbol{X^{\sigma}} = \boldsymbol{X'^{\sigma}} \ \wedge \tag{19}$$

$$\forall \boldsymbol{X^{\sigma}} \ R(\boldsymbol{X^{\sigma}}) \ \rightarrow \ \exists \boldsymbol{X} \ \sigma(\boldsymbol{X},\boldsymbol{X^{\sigma}}). \tag{20}$$

In the important case of a set of UCSs, the following specialized result holds:

Corollary 1. *Let $\psi \doteq \exists \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}} \ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}})$ be a specification, $O \in \boldsymbol{\mathcal{O}}$, and D and C a partition of its argument indexes. A set of permutations σ over $type(\underset{C}{\pi}(O))$, finitely characterized by the additional conditions encoded in a formula $\gamma(\sigma, \boldsymbol{\mathcal{T}})$, are all UCSs for ψ iff the following formula (open wrt $\boldsymbol{\mathcal{O}}, O_\tau, \boldsymbol{\mathcal{R}}, \sigma$)[4] is valid:*

$$\tau(O,O_\tau) \ \rightarrow \ \exists \boldsymbol{\mathcal{A}} \ \phi(O_1,\ldots,O,\ldots,O_n,\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{R}}) \leftrightarrow \exists \boldsymbol{\mathcal{A}} \ \phi(O_1,\ldots,O_\tau,\ldots,O_n,\boldsymbol{\mathcal{A}},\boldsymbol{\mathcal{R}}) \tag{21}$$

with τ being:

$$perm(\sigma, type(\underset{C}{\pi}(O))) \wedge \gamma(\sigma,\boldsymbol{\mathcal{T}}) \ \wedge \ \forall \boldsymbol{X_D}, \boldsymbol{X_C}, \boldsymbol{X_C^{\sigma}} \quad O_\tau(\boldsymbol{X_D}, \boldsymbol{X_C^{\sigma}}) \ \leftrightarrow \ O(\boldsymbol{X_D}, \boldsymbol{X_C}) \wedge \sigma(\boldsymbol{X_C}, \boldsymbol{X_C^{\sigma}})$$

[4] With σ being a predicate of arity $|type(\underset{C}{\pi}(O))|$.

In general, formula (21) is second-order, because of the presence of "$\exists \boldsymbol{\mathcal{A}}$", and because $\gamma(\sigma, \boldsymbol{\mathcal{T}})$ can be second-order. Of course it reduces to a first-order formula when $\boldsymbol{\mathcal{A}} = \emptyset$ and $\gamma(\sigma, \boldsymbol{\mathcal{T}})$ is also first-order.

All the UCSs described in Examples 3 and 5 can be checked in one step by using Corollary 1, with $\gamma \equiv \mathsf{true}$. As another example, let us consider a variation of Social golfer where the following constraint is added:

$$\forall P, W, G \qquad P = p_1 \wedge \mathit{Play}(P, W, G) \;\rightarrow\; G = g_1 \tag{22}$$

forcing a particular player (denoted by the constant "p_1") to play always in the same group (denoted by constant "g_1"). In the new specification, not all UCTs denoted with *1.* in Example 5 are symmetries any more. In particular, only those permutations of groups σ such that $\sigma(g_1) = g_1$ remain symmetries. The whole set of such permutations is finitely representable as $\gamma(\sigma, \boldsymbol{\mathcal{T}}) \doteq \sigma(g_1, g_1)$, thus, they can be all verified at once by using Corollary 1.

5 Symmetry Breaking

In Section 4, we showed how logically representable sets of "structural" symmetries can be checked by reasoning on the problem specification. Here we show how such knowledge can be used in order to *modify* the specification, in order to exclude from the search space (some of) the symmetrical points. Such modifications can of course be made by working only on the specification, since they will be valid whatever instance we will consider in a later stage.

Actually, several approaches to symmetry breaking have been described in Section 1. In this paper, we focus on the first one (i.e., the addition of symmetry-breaking constraints) but, differently from other works in the literature (e.g., [7, 9]), we attack this problem at the logical level of the specification.

Definition 5 (Symmetry-breaking formula). *Given a specification $\exists \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}})$, and a logical representation $\tau(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{O}}_\tau)$ of a set of symmetries, a symmetry-breaking formula for them is a closed (except for $\boldsymbol{\mathcal{O}}$ and $\boldsymbol{\mathcal{T}}$) formula $\beta(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{T}})$ –in general in second-order logic– such that the new specification*

$$\exists \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}} \quad \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \;\wedge\; \beta(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{T}})$$

satisfies the following two requirements (we call them Conditions 1 and 2):

1. *The set of transformations τ is not a set of symmetries for the new problem any more: hence, the following formula (negation of (12)), is satisfiable:*

$$\tau(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{O}}_\tau) \;\wedge\; [\exists \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \wedge \beta(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{T}}) \;\not\Rightarrow\; \exists \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}_\tau, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \wedge \beta(\boldsymbol{\mathcal{O}}_\tau, \boldsymbol{\mathcal{T}})]\,.$$

2. *Every model of $\exists \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}})$ (i.e., every solution for any input instance) can be obtained by those of $\exists \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \wedge \beta(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{T}})$ by applying transformations in τ an arbitrary number of times:*

$$\exists \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \;\models\; \exists \boldsymbol{\mathcal{O}}_\beta\ \exists \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}_\beta, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \wedge \beta(\boldsymbol{\mathcal{O}}_\beta, \boldsymbol{\mathcal{T}}) \wedge \bigvee_{i \geq 0} \tau^i(\boldsymbol{\mathcal{O}}_\beta, \boldsymbol{\mathcal{O}})$$

where $\tau^0(\boldsymbol{\mathcal{O}}_\beta, \boldsymbol{\mathcal{O}})$ *is defined as* $\boldsymbol{\mathcal{O}}_\beta \equiv \boldsymbol{\mathcal{O}}$,[5] *and* $\tau^i(\boldsymbol{\mathcal{O}}_\beta, \boldsymbol{\mathcal{O}})$ *(*$i > 0$*) as* $\exists \boldsymbol{\mathcal{O}}' \; \tau(\boldsymbol{\mathcal{O}}', \boldsymbol{\mathcal{O}}) \wedge \tau^{i-1}(\boldsymbol{\mathcal{O}}_\beta, \boldsymbol{\mathcal{O}}')$*, with* $\boldsymbol{\mathcal{O}}'$ *a fresh set of variables.*

Lemma 1 (Alternative formulation for Conditions of Definition 5). *Formula* $\beta(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{T}})$ *is symmetry-breaking for set of symmetries* $\tau(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{O}}_\tau)$ *iff it satisfies the following two, alternative, conditions:*

1. *The following formula is satisfiable:*

$$\tau(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{O}}_\tau) \; \wedge \; \exists \boldsymbol{\mathcal{A}} \; \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \; \wedge \; [\beta(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{T}}) \; \not\leftrightarrow \; \beta(\boldsymbol{\mathcal{O}}_\tau, \boldsymbol{\mathcal{T}})] \, . \tag{23}$$

2. *It holds that:*

$$\exists \boldsymbol{\mathcal{A}} \; \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \quad \models \quad \exists \boldsymbol{\mathcal{O}}_\beta \; \beta(\boldsymbol{\mathcal{O}}_\beta, \boldsymbol{\mathcal{T}}) \; \wedge \; \bigvee_{i \geq 0} \tau^i(\boldsymbol{\mathcal{O}}_\beta, \boldsymbol{\mathcal{O}}). \tag{24}$$

If $\beta(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{T}})$ respects the above conditions, we are entitled to solve the problem $\exists \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}} \; \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \wedge \beta(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{T}})$ instead of the original one $\exists \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}} \; \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}})$. In fact, Condition 1 states that formula $\beta(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{T}})$ actually breaks τ, since, by Theorem 1, transformations in τ are not all symmetries of the rewritten problem. Furthermore, Condition 2 states that every solution of $\phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}})$ can be obtained by repeatedly applying transformations in τ to some solutions of $\phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{R}}) \wedge \beta(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{T}})$. Hence, all solutions are preserved in the rewritten problem, up to symmetric ones.

It can be observed that Condition 1, even if it behaves well when τ describes a single symmetry, is quite weak when used with a set of symmetries. This is because it is enough, for a formula β, to break just one of the symmetries in τ to satisfy it. A stronger characterization of Condition 1 for the case of τ representing a set of symmetries is currently under investigation.

As for Condition 2, it is worthwhile noting that in formula (24) i ranges over the (infinite) set of positive integers. However, once the (always finite) Herbrand universe $\mathcal{H}$ has been fixed, the number of consecutive applications of τ that lead to different extensions for predicates in $\boldsymbol{\mathcal{O}}$ is always finite (even if this value actually depends on $\mathcal{H}$). Furthermore, when dealing with UCSs on guessed predicate $O \in \boldsymbol{\mathcal{O}}$, i is bound by $n!$, where n is $\left| type(\underset{C}{\pi}(O)) \right|$, and C the set of indexes where τ focuses on, since this is the maximum number of successive applications of τ that can lead to all different permutations. However, in the following we show that in many practical circumstances, either i is bound to a known value because the value for n is known (cf., e.g., Example 2), or many interesting symmetry-breaking formulae satisfy Condition 2 of Definition 5 by design, with a very low i.

We observe that breaking a symmetry is sound, i.e., it preserves at least one solution, as shown by the following theorem:

[5] In general, given two vectors of variables $\boldsymbol{X}$ and $\boldsymbol{Y}$ of the same length n, by $\boldsymbol{X} \equiv \boldsymbol{Y}$ we denote the formula $\bigwedge_{i=1}^{n} (X_i \leftrightarrow Y_i)$.

Theorem 4 (Symmetry-breaking formulae preserve satisfiability). *Let ψ, τ, and β as in Definition 5. For each input instance $\boldsymbol{\mathcal{I}}$, if $\exists \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{I}})$ has solutions, then also $\exists \boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}\ \phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{I}}) \wedge \beta(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{T}})$ has solutions.*

Example 6 (HP 2D-Protein folding (Example 4 continued)). Let us consider the following UCS τ that focuses on the output guessed predicate *Move*, with $D = \{1\}$ and $C = \{2\}$, characterized by the following permutation σ of $\{N, S, E, W\}$:

$$\sigma(N) = N,\ \sigma(S) = S,\ \sigma(E) = W,\ \sigma(W) = E \text{ (flip horizontally)}$$

The following formula $\beta^{E,W}_{\text{least}}(\mathit{Move}, \mathit{index})$ is symmetry-breaking for it:

$$\beta^{E,W}_{\text{least}} \doteq \forall I\ \mathit{index}(I) \wedge \mathit{Move}(I, W) \rightarrow \exists I'\ \mathit{index}(I') \wedge (I' \leq I) \wedge \mathit{Move}(I', E) \quad (25)$$

since it forces the protein shape to move East before moving West. Condition 1, i.e., formula (23) is satisfied by, e.g., the instance $[H, H]$, and the extension $\{\langle 1, E\rangle\}$ for *Move*. As for Condition 2, it holds even by limiting i to only 0 and 1.

A different symmetry-breaking formula for the same symmetry is:

$$\beta^{E,W}_{\leq} \doteq |\{i : \mathit{Move}(i, E)\}| \leq |\{i : \mathit{Move}(i, W)\}|, \quad (26)$$

that forces the protein "head" to move West at least the same number of times it moves East.

Example 7 (Social golfer (Example 5 continued)). Let us consider all UCSs that focus on the output guessed predicate *Play*, with $D = \{1, 2\}$ and $C = \{3\}$, i.e., all permutations of groups. The following formula (where we assume that a total ordering is given on tuples of relations in $\boldsymbol{\mathcal{T}}$, hence also on their Cartesian product) is symmetry-breaking (according to Definition 5) for all of them:

$$\begin{aligned}\beta_{\text{least}}(\mathit{Play}, \mathit{player}, \mathit{week}, \mathit{group}) \doteq\ & \forall G, G'\ \mathit{group}(G) \wedge \mathit{group}(G') \wedge (G \leq G') \rightarrow \\ & \forall P, W, P', W' \mathit{least}((P, W), G) \wedge \mathit{least}((P', W'), G') \rightarrow (P, W) \leq_{PW} (P'W')\end{aligned} \quad (27)$$

with $\leq_{PW}$ the total order derived from $\leq$ on players and weeks. It forces the group assignment to be such that, for all G, G' such that $G \leq G'$, the least pairs P, W and P', W' such that $\mathit{Play}(P, W, G)$ and $\mathit{Play}(P', W', G')$ are such that $(P, W) \leq_{PW} (P', W')$. [6] As a consequence, we have that the first player always plays in the first group. We can break other symmetries (e.g., permutations of weeks or players) in a similar way, and get the symmetry-breaking constraints described in [23].

Social golfer is well known also because it is one of the prototypical examples of problems having a 2D *matrix model* (where rows are players, columns are weeks, and entries are groups) exhibiting all row and column symmetries. For these problems, the lex^2 symmetry-breaking constraint, that forces a lexicographic

[6] $\mathit{least}((P, W), G)$ can be written in first-order logic as: $\mathit{Play}(P, W, G) \wedge \forall \overline{P}, \overline{W}\ \mathit{Play}(\overline{P}, \overline{W}, G) \rightarrow (P, W) \leq_{PW} (\overline{P}, \overline{W})$.

ordering on both rows and columns of the matrix has been proposed [9]. It is possible to show that the lex^2 symmetry-breaking constraint can be formulated in ESO by, e.g., a formula $\beta_{lex^2}(Play, player, week, group) \doteq \beta^P_{lex} \wedge \beta^W_{lex}$ where $\beta^P_{lex}(Play, player, week, group)$ is given by:

$$\begin{aligned}
\forall P, P' \quad & player(P) \wedge player(P') \wedge P < P' \ \rightarrow \\
& \exists \overline{W} \ week(\overline{W}) \ \wedge \ \forall W \ week(W) \ \wedge W < \overline{W} \ \rightarrow \\
& \quad \forall G, G' \ \big(Play(P, W, G) \wedge Play(P', W, G')\big) \ \rightarrow \ G = G' \ \wedge \\
& \quad \forall G, G' \ \big(Play(P, \overline{W}, G) \wedge Play(P', \overline{W}, G')\big) \ \rightarrow \ G < G' \ \vee \\
& \forall W, G, G' \ \big(Play(P, W, G) \wedge Play(P', W, G')\big) \ \rightarrow \ G = G'
\end{aligned}$$

that forces a lexicographic ordering among the rows of the matrix, and β^W_{lex} by a similar formula, that forces a lexicographic ordering among the columns.

It is worth noting that from the above formulae, it is straightforwardly possible to derive general schemas, that can be used to break symmetries on many different specifications. To this end, we note that formulae of the kind β_{least}, $\beta_{\leq}$, and β_{lex^2} make the right part of (24) a tautology (it is enough to consider, e.g., in the first two cases, $i \in \{0, 1\}$), and hence they are guaranteed to respect Condition 2 of Definition 5, independently on the specification constraints. This kind of schemas for βs can be used as a library, thus making a first step towards the automatic generation of guaranteed correct symmetry-breaking formulae (cf., e.g., the nature of symmetry-breaking constraints added to CSPs in [7]).

6 Experiments

In this section we show that in many cases, even if applying the technique proposed in Section 5 naively, impressive speed-ups in performances can be obtained on different problems. To this end, we show the results of the following experiments, performed with Ilog OPLSTUDIO, using state-of-the-art solvers CPLEX (a MP solver) and SOLVER (a general CP one):

- Graph k-coloring, on instances from the DIMACS repository; we broke UCSs in Example 3 with β_{least} and $\beta_{\leq}$ (using CPLEX and SOLVER);
- Social golfer, on several negative instances, with β_{lex^2} (SOLVER);
- Protein folding, by using a composition of $\beta^{E,W}_{\text{least}}$ and $\beta^{N,S}_{\text{least}}$, (SOLVER), on several benchmark instances (some of them from [12]).

Results are often good: as for k-coloring using CPLEX (cf. Table 1(a)), speed-ups up to 90% have been observed for many instances (especially when using β_{least}), even if for some others the overhead of adding such constraints leads to poorer performances (cf. also [22]). As for Social golfer instead (cf. Table 1(b)), adding β_{lex^2} leads to impressive time savings on negative instances, usually around 99%. A similar behavior has been observed for Protein folding (cf. Table 1(c)) –we solved the optimization version– with savings up to 73% (often more than 50%).

7 Conclusions

In this paper we dealt with symmetry checking and breaking at the logical level of the specification. We observed that in many cases, symmetries arise from the structure of the problem, and not from input data. Hence, from a methodological point of view, it makes sense inferring such symmetries by reasoning on the problem model. Furthermore, since specifications can be regarded as logical formulae, such tasks reduce to tautology or satisfiability checking, and hence they can be automated by computer tools (even if the general problem is undecidable). To this end, in [5] we show with several examples how first-order theorem provers and finite model finders can be effectively and efficiently used in the important case where formulae to be checked are first-order.

As for symmetry-breaking, adding constraints to the specification may, in general, lead to some overhead, and we don't exclude that, for some problems,

Table 1. Solving times (seconds) for k-coloring (CPLEX) (a), Social golfer (SOLVER) (b), and Protein folding (SOLVER) (c). '–' means that the solver did not terminate in one hour

			CPLEX				
			No s.b.	β_{least}		$\beta_<$	
Instance	k	Sol?	Time	Time	% sav.	Time	% sav.
DSJC1000.1	24	N	–	368.46	>89.77	–	–
DSJC125.5	8	N	15.32	13.21	13.77	10.51	31.40
DSJC125.5	25	Y	–	2337.29	>35.08	2177.21	39.52
DSJC125.9	21	N	1408.23	2080.21	-47.72	1088.65	22.69
DSJC250.5	10	N	–	2158.75	>40.03	2432.55	32.43
DSJC500.1	11	N	2.53	–	$-\infty$	–	$-\infty$
fpsol2.i.2	21	N	139.80	43.70	68.70	102.20	26.90
fpsol2.i.2	31	Y	–	397.61	>88.96	–	–
fpsol2.i.3	31	Y	–	330.22	>90.83	–	–
le450_25a	21	N	84.73	95.32	-12.50	46.51	45.11
le450_25a	25	Y	3536.41	–	<-1.80	1783.23	49.58
miles500	19	N	2.31	–	$-\infty$	1.67	27.71
mulsol.i.1	30	N	–	10.61	>99.71	–	–
mulsol.i.1	49	Y	–	311.12	>91.36	–	–
mulsol.i.2	30	N	–	10.98	>99.70	–	–
mulsol.i.2	31	Y	26.75	48.67	-81.94	–	$-\infty$
mulsol.i.3	30	N	–	10.78	>99.70	–	–
mulsol.i.3	31	Y	55.77	43.65	21.73	284.32	-409.81
mulsol.i.4	30	N	–	10.99	>99.69	–	–
mulsol.i.4	31	Y	47.46	14.25	69.97	–	$-\infty$
mulsol.i.5	30	N	–	11.12	>99.69	–	–
mulsol.i.5	31	Y	166.85	20.68	87.61	64.56	61.31
myciel4	4	N	5.22	0.87	83.33	8.46	-62.07

(a)

Instance				SOLVER		
Plrs	Wks	Grps	Solv?	No s.b.	β_{lex^2}	% sav.
6	6	3	N	2267.35	2.12	99.91
6	7	3	N	273.53	4.23	98.45
6	8	3	N	96.67	10.31	89.33
9	5	3	N	–	1.05	>99.97
9	6	3	N	342.24	3.86	98.87

(b)

Instance		SOLVER		
Length	Contacts	No s.b.	$\beta^{E,W}_{\text{least}} \wedge \beta^{N,S}_{\text{least}}$	% sav.
14	5	45.38	15.1	66.73
14	2	34.29	10.05	70.69
16	7	23.95	13.27	44.59
16	6	124.12	44.21	64.38
17	6	2788.05	746.97	73.21
17	6	311.78	117.68	62.26
18	8	–	1660.35	>53.88
18	4	547.84	370.38	32.39
18	9	–	1830.02	>49.17

(c)

this technique may be worse than making symmetry breaking after instantiation (cf., e.g., [22]) or during search. However, in our approach, we make a strong decoupling between symmetry detection and breaking: all techniques for symmetry-breaking need to know the symmetries of the problem, and detecting structural ones at the model level can be a common task for all of them. In particular, detected symmetries can be broken, in principle, either by adding symmetry-breaking constraints to the specification, or by instructing search algorithms to break them during search (cf. Section 1 for references). Understanding which technique is better for a given specification is topic for future work.

As for the experiments presented in Section 6, it is worth noting that our goal is not to compare specification-level versus instance-level symmetry-breaking, but to give evidence that even a naive implementation of the proposed symmetry-breaking techniques may lead to consistent time savings. In our opinion, this is a very interesting point, since the required reasoning can be effectively automated in many practical circumstances. As an example, in [5] we present experimental results on using first-order theorem provers for automating these tasks. Moreover, we recall –cf. the end of Section 5– that well-behaved symmetry-breaking templates do exist, that satisfy Condition 2 of Definition 5 by design. Hence, in many practical circumstances, only Condition 1 of Definition 5 should be checked for a given specification, and this can be done very efficiently by using a finite model finder (cf. [5]).

References

1. *Proc. of CP 2001*, v. 2239 of *LNCS*, 2001. Springer.
2. E. Börger, E. Gräedel, and Y. Gurevich. *The Classical Decision Problem*. Perspect. in Math. Logic. Springer, 1997.
3. C. A. Brown, L. Finkelstein, and P. W. Purdom. Backtrack searching in the presence of symmetry. In *Proc. of Intl. Conf. on Applied Algebra, Algebraic Algorithms and Error Correcting codes*, v. 357 of *LNCS*, pp. 99–110, 1988. Springer.
4. M. Cadoli and T. Mancini. Exploiting functional dependencies in declarative problem specifications. In *Proc. of JELIA 2004*, v. 3229 of *LNAI*, 2004. Springer.
5. M. Cadoli and T. Mancini. Using a theorem prover for reasoning on constraint problems. In *Proc. of Intl. W. on Modelling and Reformulating CSPs: Towards Systematisation and Automation, in conj. with CP 2004*, 2004.
6. M. Cadoli and A. Schaerf. Compiling problem specifications into SAT. *Artif. Intell.* To appear.
7. J. M. Crawford, M. L. Ginsberg, E. M. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *Proc. of KR'96*, pp. 148–159, 1996. Morgan Kaufmann.
8. P. Crescenzi, D. Goldman, C. H. Papadimitriou, A. Piccolboni, and M. Yannakakis. On the complexity of protein folding. *J. of Comp. Biology*, 5(3):423–466, 1998.
9. P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In *Proc. of CP 2002*, v. 2470 of *LNCS*, page 462 ff., 2002. Springer.
10. F. Focacci and M. Milano. Global cut framework for removing symmetries. In *Proc. of CP 2001* [1], pp. 77–92.

11. R. Fourer, D. M. Gay, and B. W. Kernigham. *AMPL: A Modeling Language for Mathematical Programming*. Intl. Thomson Publ., 1993.
12. W. Hart and S. Istrail. HP Benchmarks. Available at `http://www.cs.sandia.gov/tech_reports/compbio/tortilla-hp-benchmarks.html`.
13. J. Köbler, U. Schöning, and J. Torán. *The graph isomorphism problem: its computational complexity*. Birkhauser Press, 1993.
14. K. F. Lau and K. A. Dill. A lattice statistical mechanics model of the conformational and sequence spaces of proteins. *Macromolecules*, 22:3986–3997, 1989.
15. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM Trans. on Comp. Logic*. To appear.
16. B. D. McKay. *Nauty* user's guide (version 2.2). Available at `http://cs.anu.edu.au/~bdm/nauty/nug.pdf`, 2003.
17. P. Meseguer and C. Torras. Solving strategies for highly symmetric CSPs. In *Proc. of IJCAI'99*, pp. 400–405, 1999. Morgan Kaufmann.
18. P. Meseguer and C. Torras. Exploiting symmetries within constraint satisfaction search. *Artif. Intell.*, 129:133–163, 2001.
19. I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Math. and Artif. Intell.*, 25(3,4):241–273, 1999.
20. C. H. Papadimitriou. *Computational Complexity*. Addison Wesley Publ. Co., 1994.
21. J.-F. Puget. On the satisfiability of symmetrical constrained satisfaction problems. In *Proc. of ISMIS'93*, v. 689 of *LNCS*, pp. 350–361, 1993. Springer.
22. A. Ramani, F. A. Aloul, I. L. Markov, and K. A. Sakallak. Breaking instance-independent symmetries in exact graph coloring. In *Proc. of DATE 2004*, pp. 324–331, 2004. IEEE Comp. Society Press.
23. B. M. Smith. Dual model of permutation problems. In *Proc. of CP 2001* [1], pp. 615–619.
24. P. Van Hentenryck. *The OPL Optimization Programming Language*. The MIT Press, 1999.
25. P. Van Hentenryck, P. Flener, J. Pearson, and M. Ågren. Tractable symmetry breaking for CSPs with interchangeable values. In *Proc. of IJCAI 2003*, pp. 277–282, 2003. Morgan Kaufmann.
26. P. Van Hentenryck, P. Flener, J. Pearson, and M. Ågren. Compositional derivation of symmetries for constraint satisfaction. TR 2004-022, Dep. of Inf. Tech., Uppsala Univ., Uppsala, Sweden, 2004.

Approximate Model-Based Diagnosis Using Preference-Based Compilation

Gregory Provan

Computer Science Department, University College Cork, Cork, Ireland
g.provan@cs.ucc.ie

Abstract. This article introduces a technique for improving the efficiency of diagnosis through approximate compilation. We extend the approach of compiling a diagnostic model, as is done by, for example, an ATMS, to compiling an approximate model. Approximate compilation overcomes the problem of space required for the compilation being worst-case exponential in particular model parameters, such as the path-width of a model represented as a Constraint Satisfaction Problem. To address this problem, we compile the subset of most "preferred" (or most likely) diagnoses. For appropriate compilations, we show that significant reductions in space (and hence on-line inference speed) can be achieved, while retaining the ability to solve the majority of most preferred diagnostic queries. We experimentally demonstrate that such results can be obtained in real-world problems.

1 Objective

One of the most influential approaches to model-based diagnosis (MBD) consists of compiling the diagnostic model into a representation, Θ, from which diagnoses can be more efficiently computed. This approach has been adopted within a number of approaches, e.g., [1, 2, 3]. The advantage of this approach is that the computational task is linear in the size of the compiled representation. However, the disadvantage with compiling a large model is the space required for the compilation; for example, for a model represented as a Constraint Satisfaction Problem (CSP), e.g., as a causal network [4], this space is worst-case exponential in the path-width of the CSP [5]. For real-world problems (which have large path-width or thousands of variables), the size of the compiled representation is typically too large for practical inference.

To address the large size of a compiled diagnostic model Θ, we compile a subset of the space of diagnoses, namely the most preferred subset of diagnoses, using a valuation function to specify the most preferred diagnoses. The most common valuation function is the likelihood of a fault, which can be specified in terms of a probability (e.g., [6]) or order-of-magnitude probability (e.g., [4]) assigned to failure modes. We address two well-known diagnostic compilation approaches for which valuations can be assigned to each compiled diagnosis, prime implicants [1] and consequences in d-NNF [2]. We are interested in the tradeoff between the proportion of the most-preferred diagnoses represented in a partial compilation Θ_φ versus the space saved by Θ_φ, relative to that of Θ. We use two measures to analyse this tradeoff for a partial compilation: (1) χ

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 182–193, 2005.

measures the relative fraction of important diagnoses that are generated by the partial compilation Θ_φ, relative to the space of the full compilation; and (2) λ measures the proportion of the space that a partial compilation Θ_φ requires, relative to the space of Θ.[1] We provide a theoretical bound that can be used to predict the tradeoff parameters (χ, λ) for a partial compilation, and show experimental results that such bounds are relevant for real-world problems.

We frame our analysis using the general diagnostic framework of constraint optimisation using CSPs [7]. This framework describes the diagnostic model using the CSP framework, with valuations over the CSP described using a c-semiring. For the class of CSPs we have addressed, our partial compilation results are encouraging. For partial compilations in which all failure modes are unlikely or in which some failure modes are much more likely (preferred) than others, we can produce order-of-magnitude space savings, with little loss of deductive coverage; in other words, we can have compilations with χ close to 1 and $R \ll 1$. Under these scenarios, the most likely diagnoses comprise a small fraction of the number of total diagnoses, with the majority of remaining diagnoses being significantly less likely.

This article makes two main contributions. First, it describes a general framework for MBD in which a variety of valuations and compilation techniques can be adopted. Second, it describes the conditions under which approximate preference-based compilation can significantly speed up diagnostic inference with little loss of diagnostics coverage.

2 Notation and Representation

This section introduces our notation for CSPs, for compilation, and for valuations of solutions to CSPs.

2.1 CSP Problem Formulation

We assume the CSP diagnostic formulation of [7]:

Definition 1 (Constraint Satisfaction Problem (CSP)). *A Constraint Satisfaction Problem (CSP) $\Pi = \langle \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{H} \rangle$ over $\{\top, \bot\}$ consists of:*

- *a set of variables $\mathcal{X} = \{x_1, ..., x_n\}$;*
- *for each variable x_i, a finite set $\mathcal{D}_i$ of possible values (its domain);*
- *and a set $\mathcal{C}$ of constraints restricting the values that the variables can simultaneously take. A constraint c_i is a relation defined on a subset X' of the variables, that is, $c_i \subseteq \times_j \{x_j : x_j \in X'\}$.*

The constraints c_j can be considered as functions defined over the variables in c_j, $V(c_j)$, where allowed tuples have value $\top$ and disallowed tuples have value $\bot$.

[1] λ provides a measure of the relative complexity of approximate compiled inference versus using the full model.

Diagnostic applications typically consider the case where (1) we assume a subset of distinguished unary constraints $\mathcal{H} \subseteq \mathcal{C}$ referred to as assumptions, and (2) we can measure a set $O \subseteq \mathcal{X}$ of variables, called observables. Given this framework, we can specify a diagnosis as follows:

Definition 2 (Diagnosis). *Let $\Pi = \langle \mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{H} \rangle$ be a CSP and O an observation, i.e., a constraint on variables in $\mathcal{X}$. A diagnosis of O on Π is a subset of constraints $E \subset \mathcal{H}$ such that $\mathcal{C} \cup O \cup E \not\models \perp$, or equivalently, there exists an assignment of values in $\mathcal{D}$ to $\mathcal{X}$ consistent with $\mathcal{C}$ and O.*

2.2 Valuation

In this article, we assign a valuation to unary constraints (i.e., assumptions), and then use this valuation to compute most preferred diagnoses. A valuation denotes the importance of a constraint. We represent a valuation of a constraint c using $\upsilon(c)$. Hence, we have a weighted-pair (c_i, υ_i) for each constraint and valuation. We formalise this general notion of valuation in terms of c-semiring operations [8]. Note that this formalism covers, among others, the probabilistic valuation of [6] and the order-of-magnitude probabilistic valuation of [4].

Definition 3 (c-semiring). *A c-semiring is a tuple $(A, +, \times, 0, 1)$ such that*

- *A is a set and $\{0, 1\} \in A$;*
- *$+$ is a commutative, associative and idempotent operation with unit element 0 and absorbing element 1;*
- *$\times$ is a commutative, associative and idempotent operation with unit element 1 and absorbing element 0;*
- *$\times$ distributes over +.*

Definition 4 (c-semiring constraint system). *A constraint system over a c-semiring is a constraint system where the constraints $c_j \in \mathcal{C}$ are functions defined over the variables in c_j assigning to each tuple a value in A.*

Definition 5. *An objective function υ maps tuples $Z \subseteq \mathcal{X}$ to a set A with a partial order $\preceq_A$ that forms a complete lattice.*

In the probabilistic case [6] (see Section 3.2), A is the interval $[0, 1]$ with total order $\leq$, and υ associates a probability with each failure mode assignment.

We can define an optimization task over a constraint system in terms of c-semiring operations provided that the objective function is $\times$-separable.

2.3 Compilation

Diagnosis can be formalised as a type of Consistency Maintenance Algorithm, and a number of techniques have been developed for compiling this type of task. These techniques include prime implicates [1], d-NNF [2], OBDD [9], and cluster-trees [10].

Given a set $\mathcal{C}$ of constraints, we compile the constraints after partitioning them into a constant part $\mathcal{C}_c$ and a varying part $\mathcal{C}_v$. The constant part is then replaced by an equivalent, but computationally more efficient, compiled representation $\mathcal{C}'_c$. Thus given an

entailment problem for determining consequences α of $\mathcal{C}$, i.e., $\mathcal{C}_c \cup \mathcal{C}_v \models \alpha$, we can compile $\mathcal{C}_c$ into C'_c and express this as

$$\mathcal{C}'_c \models \alpha \vee \bigvee_{\xi \in \mathcal{C}_v} \neg\xi.$$

Prior approximate compilation techniques typically weaken the problem representation. For example, papers by Selman and Kautz [11] and by del Val [12] have studied approximating propositional and First-Order formulae by Horn lowest upper bound (LUB) representations, as well as their generalisations.

In contrast to this approach, we are interested in using the prior valuations on assumptions to compile a subset of *most preferred* potential diagnoses. This is similar to the penalty logic framework introduced in [13], except that in this case we compile only a *subset* of the most preferred diagnoses, rather than the full set of ranked diagnoses. We compile the least-cost diagnoses to $\mathcal{C} \cup \mathcal{H}$ up to a threshold φ. In other words, we compile all diagnoses such that $\upsilon(E) \leq \varphi$. This approach is a general one, and can be applied to any compilation method. For example, with regard to the prime implicants (or labels) computed by an ATMS [1] or consequence generation [2], we ensure that no label (consequence) will have cost more than a bound φ.

3 Valuation-Based Diagnosis

We now introduce some well-known methods for valuations, and in later sections we will see the types of results that are possible given those valuations. We derive some theoretical results about such partial compilations, and then present experimental results for real-world models.

3.1 Valuation 1: Unary Integral Valuation

We first examine the valuation addressed in [4]. The valuation corresponds to a semiring $S_{\mathbb{N}}$ given by $\langle \mathbb{N} \cup \{\infty\}, min, +, \infty, 0\rangle$. This valuation, $\upsilon : \mathcal{H} \rightarrow \mathbb{N}^+$, is assigned to the assumptions, and is a totally ordered mapping over an diagnosis $E \subseteq \mathcal{H}$ such that the valuation for any diagnosis E is given by

$$\upsilon(E) = \sum_{H \in \mathcal{H}} \upsilon(H).$$

In other words, the valuation is a measure assigned to the assumptions (constraints) contained in E; i.e., it represents the likelihood of occurrence of the diagnosis E. Under this valuation, a 0-cost represents a normal system and increasing costs (greater than zero) correspond to increasingly unlikely (less-preferred) diagnoses. Hence, our inference objective is to compute minimum-cost diagnoses.

3.2 Valuation 2: Probabilistic Valuation

We now outline a valuation widely used in diagnosis [3, 6] and other areas of cost-based abduction. In this valuation we assign a probability p to each assumption: $Pr : \mathcal{H} \rightarrow [0, 1]$. The valuation of a diagnosis $E \subseteq \mathcal{H}$ is given by

$$Pr(E) = \prod_{H \in \mathcal{H}} \Pr(H),$$

where we assume that all assumptions are independent, such that we can compute the joint probability $Pr(E)$ by the products of the probabilities for $H \in \mathcal{H}$. The valuation corresponds to a semiring S_{Pr} given by $\langle [0,1], max, \cdot, 0, 1 \rangle$.

The semantics of this valuation are slightly different than those of Valuation 1. Starting from a maximum valuation of 1 (which represents a normal system), all valuations less than 1 correspond to solutions which are increasingly less likely (preferred). Hence, our objective is to compute *maximum*-probability diagnoses.

4 Valuations for Compilations

This section examines the valuations for compilations, and in particular looks at the trade-offs of relative size of the compilation versus the total relative value of the compilation.

We pose an optimisation task for Valuation 1, that of computing the least-cost diagnoses, and then the compile the least-cost diagnoses up to a threshold $\varphi \in \mathbb{N}^+$. In other words, we compile all diagnoses $E \in \mathcal{E}^*$ such that $v(E) \leq \varphi$.

4.1 Relative Value of a Partial Compilation

The objective of our approximate compilation is to provide coverage for a fixed percentage of possible diagnosis queries. We use the following notation for specifying the relative value of a partial compilation:

Definition 6 (Constraint Set Valuation). *The valuation associated with a constraint set $\mathcal{H}$ (or equivalently, with a complete compilation Θ of $\mathcal{H}$), is given by the sum over all valuations:*

$$v(\Theta) = \sum_{E \in 2^{\mathcal{H}}} v(E).$$

Definition 7 (Partial Constraint Set Valuation). *The valuation of a partial compilation Θ_φ with valuation threshold φ is given by*

$$v(\Theta_\varphi) = \sum_{E \in 2^{\mathcal{H}}} \{v(E) | v(E) \leq \varphi\}.$$

We use these notions to define a key parameter for our experimental analysis, the valuation coverage ratio.

Definition 8 (Valuation Coverage Ratio). *We define the* valuation coverage ratio χ *of a partial compilation Θ_φ, with valuation threshold φ, as the fraction of the complete system valuation provided by Θ:*

$$\chi = \frac{v(\Theta_\varphi)}{v(\Theta)}. \tag{1}$$

Our approach cannot use a valuation with an unbounded maximum value, e.g., ∞ for Valuation 1, as we are interested in computing ratios of cumulative valuations. For such valuations we must construct an inverse valuation, which we call a loss function, in order to compute an appropriate measure for χ. We adopt a standard decision-theoretic loss function $\mathcal{L}$, defined for constraint c as $\mathcal{L} : v(c) \rightarrow [0, 1]$.[2] Using a loss function, we can modify Equation 1 into: $\chi = \frac{\mathcal{L}(\Theta_\varphi)}{\mathcal{L}(\Theta)}$. We say that a partial compilation is *effective* if it has a high coverage ratio.

There are a variety of methods that we can use to map valuations with unbounded maximum values into a loss function. For example, we can define two classes of loss function for Valuation 1 as follows. First, we could adopt an appropriate *log* transformation of the form: $v(e) \rightarrow log_\zeta(\zeta - v(e))$,[3] for a set of diagnoses with maximum valuation ζ. A second method of representing a loss function uses a parameterised equation of the form: $\mathcal{L}(c) = \gamma\epsilon^{v(c)}$, for constant γ and parameter ϵ.[4]

4.2 Relative Memory of a Partial Compilation

The second key parameter in which we are interested is the relative memory of a partial compilation, which we can define as follows. Let $|\Theta|$ be a measure for the size of the original compiled CSP, and $|\Theta_\varphi|$ be a measure for the size of the CSP compiled based on valuation threshold φ. For simplicity, we assume that all diagnoses (solutions) take up equal memory, and define a ratio based only on the relative number of solutions.

Definition 9 (Memory Reduction Factor). *The memory reduction of partial compilation, with respect to compiling the full CSP, is given by* $\lambda = \frac{|\Theta_\varphi|}{|\Theta|}$.

4.3 Analysis of Different Valuations

This section analyses the impact of two parameters on the size and effectiveness of a partial compilation: (1) *valuation distribution*, the relative proportion of different preferences; and (2) *valuation differential*, the difference in degree of preference between any two different valuations. A model may specify a preference ordering in which some assumptions are very strongly preferred, and others are not preferred; in that case the distribution specifies the *relative proportions* of highly preferred to not preferred assumptions, and the differential indicates the difference in *degree* of preference among the assumption valuations.

Example 1. Consider a simple example with a lattice defined over assumptions $\{a, b, c, d\}$ *and semiring* S_{Pr}. *In this case we assign probabilities describing the failure likelihoods of the variables. If we have a valuation distribution given by* $Pr(a) = Pr(b) = 0.09$, *and* $Pr(c) = Pr(d) = 0.01$,[5] *then assume that we have a model in*

[2] Note that Valuation 2 automatically satisfies this requirement, but Valuation 1 does not.

[3] Note that a complete mapping is more complicated than this example.

[4] This mapping corresponds to the semiring $S_\mathcal{L}$ given by $\langle [0, 1], max, \cdot, 0, 1\rangle$, and is very close to the calculus proposed in [14].

[5] Note that there is roughly an order-of-magnitude differential between a strong preference, e.g., $Pr(a)$, and a weak preference, e.g., $Pr(c)$.

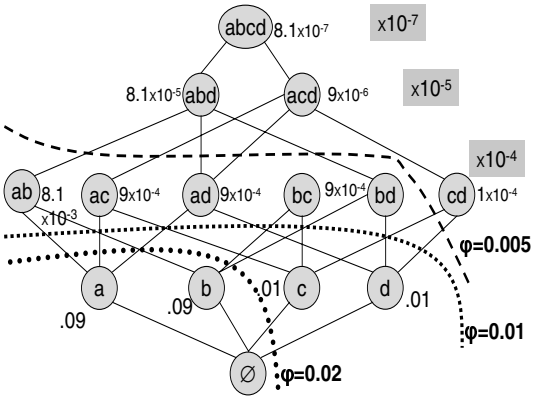


Fig. 1. Lattice for simple example, showing the probabilistic valuations to lattice elements, and the lattice elements (in lower left corner) with probability greater than 0.2, 0.1 and 0.05

which a full compilation would generate a lattice containing the 13 diagnoses shown in Figure 1.[6] *The lattice elements represent all possible consistent diagnoses for potential observations; the valuations on the diagnoses can be used to rank-order the diagnoses in terms of likelihood; for example, if we have diagnosis set* $\{cd \vee abd\}$ *for an observation O, the most-likely diagnosis will be* $\{cd\}$.

Consider the effect of introducing partial compilations with bound φ. *If we introduce a bound of* $\varphi = 0.02$, *then we would compile only 2 diagnoses (*$\{a, b\}$*) out of 13 total diagnoses; note that this would give valuation coverage ratio* χ *of 0.849, and relative memory* λ *of 0.15. In this case, we have used only 15% of the memory of the total compilation, and can answer roughly 85% of the diagnosis queries. Decreasing* φ *to 0.01 and 0.005 results in* (χ, λ) *pairs of (.943, .308) and (.999, .769) respectively. These results show that we obtain diminishing increases in* χ *as we compile more diagnoses.*

If we increase the differential in this valuation to roughly two orders of magnitude, i.e., $Pr(a) = Pr(b) = 0.09$, *and* $Pr(c) = Pr(d) = 0.001$, *then the relative coverage of an equivalent partial compilation increases significantly. For* $\varphi = 0.02$ *we obtain a* (χ, λ) *pair of (0.992, 0.15), in which 15% of the memory covers over* **99**% *of the most-preferred diagnosis queries.* □

We can study the impact of valuation differential on partial compilation tradeoffs for Valuation 1 using the loss function $\mathcal{L}(c) = \gamma\epsilon^{v(c)}$, for $\epsilon \leq 1$. Figure 2 shows the impact of the value of ϵ on the types of tradeoff curves that are possible. At one extreme, the value $\epsilon = 1$ produces an equi-loss situation where we generate a flat lattice that does not even respect subset inclusion; hence, there is no value to compilation. The benefit of compilation improves as ϵ grows smaller, i.e., as the gap between different valuations increases.

In our analysis, we have found that the relative efficiency of a partial compilation, i.e., having a high query coverage with large reduction in memory, is directly related

[6] The lattice elements $\{abc\}$ and $\{bcd\}$ have been ruled inconsistent by the constraints.

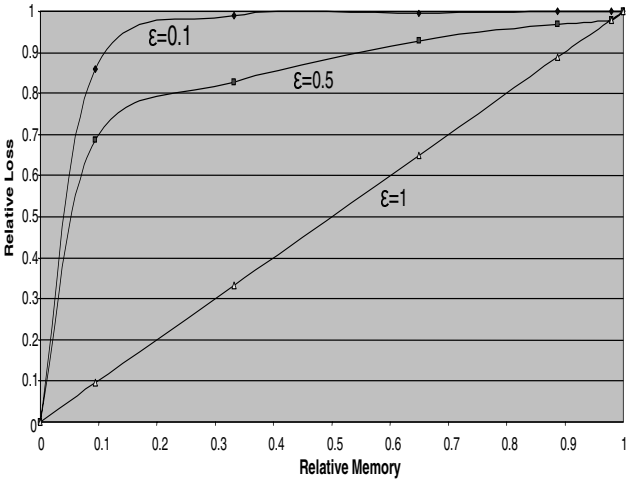


Fig. 2. Curves depicting the influence of the value of ϵ on the tradeoff curves

to the type of valuation. If a valuation is skewed, in the sense that some diagnoses are highly preferred and others are not at all preferred, then we can compute a very efficient partial compilation. If most diagnoses are relatively equally preferred, then little is gained by partial compilations.

Although it is too complicated to derive closed-form representations for the valuation tradeoff space in the general case, we can derive results for simple cases. For example, we can use the following result as a bound on the impact of partial compilation:[7]

Lemma 1. *Consider a model with n variables, where each diagnosis has identical loss of $l \in (0, 1]$. We generate a partial compilation, based on the maximum number q of variables in any diagnosis, with parameters given by:*

$$\chi = \frac{\sum_{i=1}^{q} \binom{n}{i} l^i}{(1+l)^n - 1}; \quad \lambda = \frac{\sum_{i=1}^{q} \binom{n}{i}}{2^n - 1}. \tag{2}$$

Equation 2 predicts a series of curves similar to those shown in Figure 2 for $\epsilon < 1$. Each predicted curve defines an upper bound for the expected (χ, λ) results of a partial compilation, i.e., it specifies the effectiveness of the compilation in the best case. We now present experimental results for real-world examples that show that the predictions of Equation 2 are relevant to real-world application problems.

5 Experimental Analysis

We have performed a set of empirical studies of compilation coverage. We represented the diagnostic models as causal networks [2], which is a CSP representation with propo-

[7] If we start with valuation (as in Valuation 1) that generates an inverse χ, then we must map this into a loss function using an approach such as that described in Section 4.1.

sitional constraints and non-negative integer weights similar to Valuation 1. We then generated complete DNNF-compilations of all models, and used various thresholds φ to compute partial compilations from these. For each pair comprising a full and partial compilation, we posed identical queries, and compared the statistics of correct query-responses for the partial and full compilations. This section describes these studies, focusing first on the structural parameters, and then on the weights.

5.1 Structural Parameters

We have performed experiments on a collection of real-world models, including models of hydraulic, electrical and mechanical systems. Table 1 shows the basic parameters of the models we used for experimentation. The model classes are as follows: (a) the AC-v1 through AC-v4 models are for multiple aircraft subsystems; (b) the MCP models are for a control system. Each model class has multiple models, denoted by version numbers (e.g., v2), which denote the increased size and complexity over the basic model (v1).

Table 1 shows data for a variety of models. One of the key factors to note is the memory required for the compiled model, displayed in the last column. In particular, the models cover memory values ranging from small (20.1KB) to large (52.4MB). As noted earlier, the memory of the compiled model is our metric for evaluation complexity, since evaluating a model is linear in the size of the compiled data. As a consequence, it is important to note that models with compiled data in excess of 20-30MB are computationally expensive to evaluate.

Model Size Parameters. We have performed experiments to study the dependence of compiled-model performance on the parameters $\mathcal{C}$ and $\mathcal{H}$. Figure 3 shows the coverage versus relative memory for four different aircraft sub-system models of increasing size, AC-v1 through AC-v4. All models have identical failure-mode probabilities of 0.05. Note that every such coverage/memory curve has a similar shape, with the coverage asymptotically approaching 1 as memory increases. This particular graph shows how the curves are displaced downwards (meaning reduced coverage for any relative memory value) as the models grow in size and complexity.

Table 1. Model Statistics for Real-world Models. We report data for the total number V of variables, number $\mathcal{H}$ of assumables, number O of observables, and memory for the full compiled model

Name	V	$\mathcal{H}$	O	Memory (KB)
AC-v1	12	5	8	1439
AC-v2	41	9	13	37,626
AC-v3	64	17	12	52,376
AC-v4	70	19	13	52,447
MCP	40	20	5	20.1
MCP-extended-v1	66	32	5	22.5
MCP-extended-v2	66	32	8	359.9

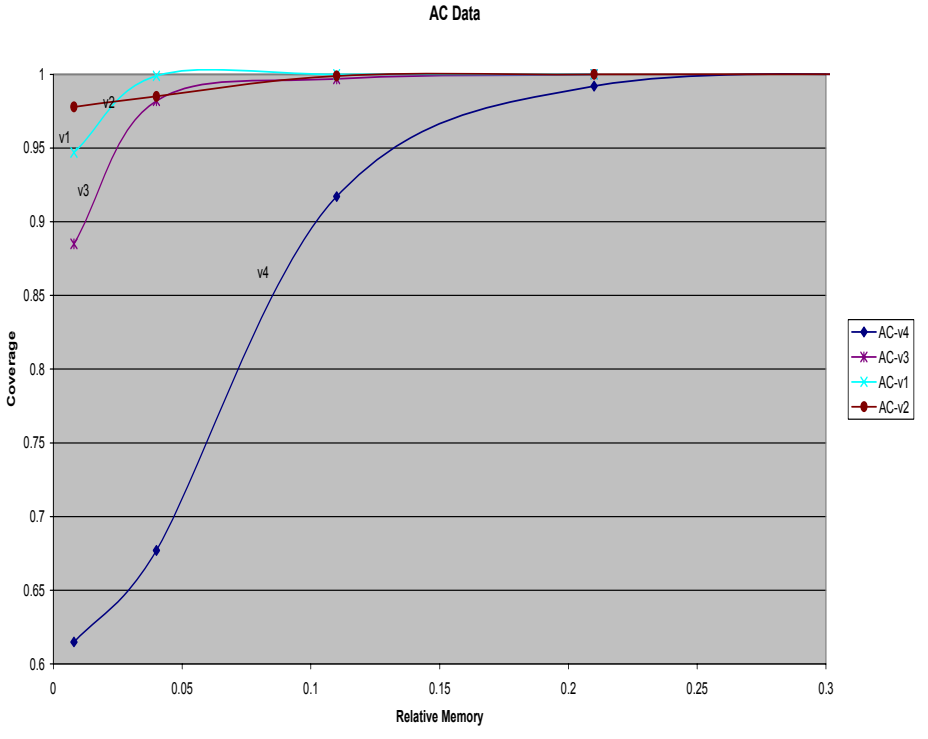


Fig. 3. Graph showing tradeoff of coverage versus relative memory for four different aircraft sub-system models, AC-v1 through AC-v4

5.2 Valuations

We have performed a variety of experiments to study the influence of assumption probability on coverage. In these experiments, we assigned different probability values to the assumptions, and report our results using the mean probability, averaged over all probabilities assigned to failure-states in $\mathcal{H}$.

Figure 4 shows the effect of mean assumption probability values on the coverage for two control models, a basic model and an extension of that model. These figures show how the mean probability value reduces the coverage values. For example, Figure 4(b) shows that for small probability values, the coverage asymptotes very quickly to coverage values near 1 at relatively small relative memory values, but as the mean probability gets larger, greater memory is needed to achieve high coverage values. These experimental results concur relatively well with predictions made by equation 2.

This experimental analysis has shown that Equation 2 provides bounds that can predict results such as:

- For a given CSP size, it tells you the type of skewed loss function that guarantees an efficient partial compilation.
- Given an appropriate loss function, it can tell you the coverage value χ that is achievable.

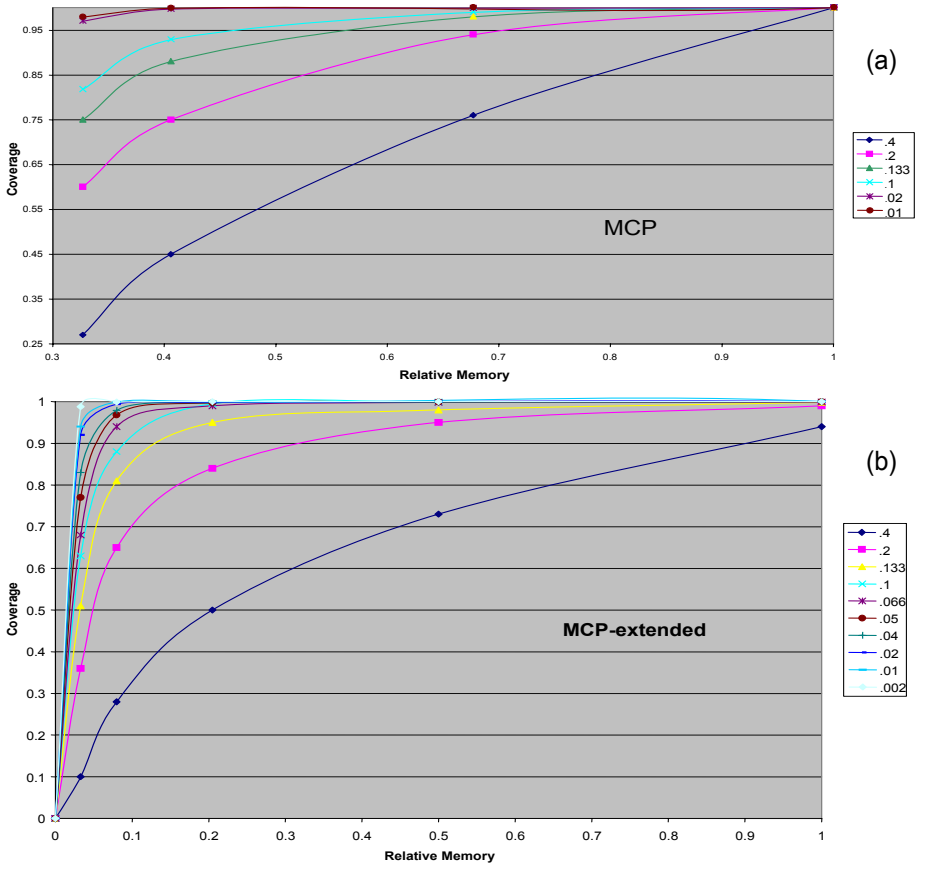


Fig. 4. Graph showing tradeoff of coverage versus relative memory for control sub-system model, for various mean loss values

– Given an appropriate loss function and required coverage value χ^*, it can predict the size of the partial compilation.

6 Summary

The article described a partial-compilation technique for improving the efficiency of model-based diagnosis, and more generally for any compilation-based inference with preferences. For DNNF-compilations, we showed that significant reductions in space (and hence on-line inference speed) can be achieved, while retaining the ability to solve the majority of diagnosis queries. We experimentally demonstrated that such results can be obtained in real-world problems. For example, under skewed preference structures, we have found that extremely good coverage can be provided by relatively small partial compilations. Given the general c-semiring CSP framework for compilation, we argue that this partial compilation approach will work for a variety of c-semiring CSPs, and

for compilation methods in which valuations can be assigned to compiled diagnoses (e.g., ATMS labels).

These results imply that high-reliability systems need only a relatively small compiled model to guarantee high diagnostic coverage. In contrast, low-reliability systems need a relatively large compiled model, e.g., containing up to 10 simultaneous faults (depending on the system), to guarantee high diagnostic coverage.

This analysis needs to be extended to cover failure consequences, i.e., incorporate utility functions. For many applications, e.g., commercial aircraft and space shuttle missions, one is interested in the low-probability/high-consequence failures. Our future work plans to analyse such situations.

References

1. de Kleer, J.: An Assumption-based TMS. AI Journal **28** (1986) 127–162
2. Darwiche, A.: A compiler for deterministic, decomposable negation normal form. In: Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI), Menlo Park, California, AAAI Press (2002) 627–634
3. Console, L., Portinale, L., Dupre, D.T.: Using Compiled Knowledge to Guide and Focus Abductive Diagnosis. IEEE Trans. on Knowledge and Data Engineering **8(5)** (1996) 690–706
4. Darwiche, A.: Model-based diagnosis using structured system descriptions. Journal of Artificial Intelligence Research **8** (1998) 165–222
5. Bodlander, H.: Treewidth: Algorithmic techniques and results. In: Proc. 22nd Intl. Symp. on Mathematical Foundations of Computer Science, MFCS'97. Volume 1295 of Lecture Notes in Computer Science. Springer-Verlag (1997) 29–36
6. de Kleer, J.: Focusing on Probable Diagnoses. In: Proc. AAAI. (1991) 842–848
7. Sachenbacher, M., Williams, B.: Diagnosis as semiring-based constraint optimization. In: Proceedings of ECAI'04, Valencia, Spain (2004)
8. Bistarelli, S., Montanari, U., Rossi, F.: Semiring-based constraint logic programming: syntax and semantics. ACM TOPLAS **23** (2002)
9. Bryant, R.E., Meinel, C.: Ordered binary decision diagrams. In Hassoun, S., Sasao, T., eds.: Logic Synthesis and Verification. Kluwer Academic Publishers (2001)
10. Pargamin, B.: Extending Cluster Tree Compilation with non-Boolean Variables in Product Configuration. In: IJCAI. (2003)
11. Selman, B., Kautz, H.: Knowledge compilation and theory approximation. Journal of the ACM **43** (1996) 193–224
12. del Val, A.: An analysis of approximate knowledge compilation. In: Proc. IJCAI. (1995) 830–836
13. Darwiche, A., Marquis, P.: Compilation of weighted propositional knowledge bases. In: Proceedings of the Workshop on Nonmonotonic Reasoning, Toulouse, France (2002)
14. Spohn, W.: Ordinal conditional functions: A dynamic theory of epistemic states. In Harper, W.L., Skyrms, B., eds.: Causation in Decision, Belief Change, and Statistics. Reidel, Dordrecht, Netherlands (1988) 105–134

Function Approximation via Tile Coding: Automating Parameter Choice

Alexander A. Sherstov and Peter Stone

Department of Computer Sciences,
The University of Texas at Austin,
Austin, TX 78712 USA
{sherstov, pstone}@cs.utexas.edu

Abstract. Reinforcement learning (RL) is a powerful abstraction of sequential decision making that has an established theoretical foundation and has proven effective in a variety of small, simulated domains. The success of RL on real-world problems with large, often continuous state and action spaces hinges on effective *function approximation.* Of the many function approximation schemes proposed, *tile coding* strikes an empirically successful balance among representational power, computational cost, and ease of use and has been widely adopted in recent RL work. This paper demonstrates that the performance of tile coding is quite sensitive to parameterization. We present detailed experiments that isolate the effects of parameter choices and provide guidance to their setting. We further illustrate that *no single parameterization* achieves the best performance throughout the learning curve, and contribute an *automated* technique for adjusting tile-coding parameters online. Our experimental findings confirm the superiority of adaptive parameterization to fixed settings. This work aims to automate the choice of approximation scheme not only on a problem basis but also throughout the learning process, eliminating the need for a substantial tuning effort.

1 Introduction

Temporal-difference reinforcement learning (RL) is a powerful machine-learning methodology that has an established theoretical foundation and has proven effective in a variety of small, simulated domains. The application of RL to practical problems, however, is problematic due to their large, often continuous state-action spaces. Recently RL has been successfully applied to larger problems, including domains with continuous state-action spaces. The success of RL in such cases critically depends on effective *function approximation*, a facility for representing the value function concisely at infinitely many points and generalizing value estimates to unseen regions of the state-action space.

A variety of function approximation methods for RL have been proposed, including simple discretization, radial basis functions, instance- and case-based approximators, and neural networks [1]. These methods trade off representational power, computational cost, and ease of use. *Tile coding* [2] is a linear function-approximation method that strikes an empirically successful balance along these dimensions and has been widely adopted in recent work [3, 1, 4, 5, 6]. The success of tile coding in practice depends in

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 194–205, 2005.

large part on parameter choices. We are not aware of any detailed studies of the effects of parameters in tile coding, an omission we set out to address.

This paper makes two chief contributions. First, we present a controlled empirical study of the effects of parameters in tile coding. While it is natural to expect the right parameterization to depend on the *problem* at hand, we additionally demonstrate that no single parameterization achieves the best performance on the *same* problem throughout the learning curve. Our analysis isolates the causes of these phenomena. Second, this paper contributes an automated scheme for adjusting tile-coding parameters online. We demonstrate the superiority of online parameter adjustment to any fixed setting.

Our work on adaptive parameterization in tile coding automates the choice of an appropriate approximation scheme for any given RL *problem* and *learning stage*. The designer need only specify a parameter range, leaving it up to the algorithm to determine the right settings throughout execution. Viewed differently, our work unifies fixed approximation schemes into a more powerful and generic scheme. We validate our insights empirically in the context of RL, arguably the most realistic and successful abstraction of sequential decision making to date.

The remainder of this paper is organized as follows. Section 2 provides a brief overview of RL and describes tile coding and our testbed domain. Experimental results in multiple learning settings and an accompanying analysis are presented in Sections 3 and 4, respectively. Section 5 builds on those findings to propose an automated parameter-adjustment scheme and demonstrates its effectiveness empirically. Section 6 concludes with a summary.

2 Preliminaries

This section introduces reinforcement learning (RL) and tile coding and describes the testbed domain used in our experiments.

2.1 Reinforcement Learning

In RL [2], a *learner* is placed in a poorly understood, possibly stochastic and non-stationary *environment*. The learner interacts with the environment at discrete time steps. At every time step, the learner can observe and change the environment's *state* through its *actions*. In addition to state changes, the environment responds to the learner's actions with a *reward*, a scalar quantity that represents the immediate utility of taking a given action in a given state. The learner's objective is to develop a *policy* (a mapping from states to actions) that maximizes its long-term return.

Formally, an RL problem is given by the quadruple $\langle \mathcal{S}, \mathcal{A}, t, r \rangle$, where $\mathcal{S}$ is a finite set of states; $\mathcal{A}$ is a finite set of actions; $t : \mathcal{S} \times \mathcal{A} \to \Pr(\mathcal{S})$ is a *transition function* that specifies the probability of observing a certain state after taking a given action in a given state; and $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a *reward function* that specifies the expected reward upon taking a given action in a given state. Given a stream of rewards $r_0, r_1, r_2, \ldots$, the associated return is defined as $\sum_{i=0}^{\infty} \gamma^i r_i$, where $0 \leq \gamma \leq 1$ is the *discount factor*. The learner experiences the world as a sequence of states, actions, and rewards, with no prior knowledge of the functions t and r. A practical vehicle for learning in this setting is the

value function $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that yields the expected long-term return obtained by taking a certain action in a given state and following policy π thereafter. The widely used Q-learning algorithm [7] maintains and iteratively updates an approximation to the Q-value function of the optimal policy.

2.2 Tile Coding

In practical applications of RL, states and actions are defined by continuous parameters such as distances and voltages. As a result, the sets $\mathcal{S}$ and $\mathcal{A}$ are typically large or infinite, and learning the value function requires some form of *function approximation*. In *tile coding*, the variable space is partitioned into *tiles*. Any such partition is called a *tiling*. The method uses several overlapping tilings and for each tiling, maintains the weights of its tiles. The approximate value of a given point is found by summing the weights of the tiles, one per tiling, in which it is contained. Given a training example, the method adjusts the weights of the involved tiles by the same amount to reduce the error on the example.

Figure 1 illustrates tile coding as it is used in this paper. The variable space consists of a single continuous variable x. The tiles are all the same width and adjacent tilings are offset by the same amount, the type of tiling organization we refer to as *canonical*. Figure 1 also illustrates the computation of value estimates. A tiling organization such as those in Figure 1 is given by tile width w and the number of tilings t. The ratio w/t is the *resolution* of a tiling organization. Speaking of tiling organizations that provide the same resolution, we refer to the number of tilings as the *breadth of generalization* since tiling organizations with more tilings generalize more broadly. This happens because the span of the tiles activated by an update grows with the number of tilings. A degenerate form of tile coding is straight discretization (the organization with a single tiling in Figure 1), which does not generalize across tile boundaries.

Note that tile coding is a *piecewise constant* approximation scheme: for any assignment of the tile weights, there will be actions within resolution r of each other that map to the same set of tiles and share the same value estimate. When pondering an action choice in this setting, our RL algorithm picks the middle action. While tile coding does not support *truly* continuous learning, its generalization capability makes it far superior to straight discretization. Finer distinctions can always be learned by increasing the resolution r. An initial result regarding tiling organizations is (a proof sketch is in the appendix):

Theorem 1. *For every $m, n \geq 1$, the sets of functions representable by m- and n-tiling canonical univariate organizations with the same resolution are identical.*

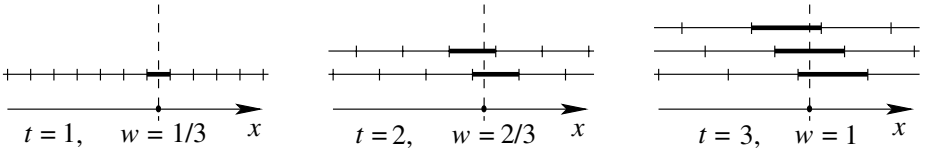


Fig. 1. One-, two-, and three-tiling canonical organizations with the same resolution $r = 1/3$. The number of tilings t and tile width w are specified for each organization. In each case, the weights of the highlighted tiles are summed to obtain the value estimate for the indicated point

For example, the three organizations in Figure 1 are functionally equivalent. Despite this representational equivalence and identical *asymptotic* performance, tiling organizations with more tilings generalize more broadly and perform differently on RL tasks. This paper assumes a fixed resolution r and studies the role of generalization breadth t as a parameter. Our work seeks to identify how varying the breadth of generalization—while preserving representational equivalence—affects performance.

2.3 Testbed Domain

Our testbed domain is a grid world, shown along with an optimal policy in Figure 2. Two locations of the grid world are designated as "start" and "goal," with the learner's objective being to navigate from the start cell to the goal cell. Another type of cell is a wall that the learner cannot pass through. Finally, certain cells are designated as an abyss. This grid world task is episodic, ending with the learner falling into the abyss ("stepping off the cliff") or successfully entering the goal state. The state variables are the cell coordinates x and y (the start state is at the origin).

The learner's actions are of the form (d, p), where $d \in \{\text{NORTH, SOUTH, EAST, WEST}\}$ is an intended direction of travel and p is a real-valued number between 0 and 1. The learner moves in the requested direction with probability $F(x, y, p)$, and in one of the three other directions with probability $(1 - F(x, y, p))/3$. Moves into walls and off the edge of the grid world result in no change of cell. F is a cell-dependent function that maps p to $[0.5, 1]$. The two "extreme" F functions are shown in Figure 3, and the F functions for all other cells are successive interpolations between these two.[1] This design of F was intended to ensure continuity as well as multiple local maxima and minima. To illustrate, consider choosing an action in a cell governed by the solid F curve in Figure 3. A choice of $p \approx 0.78$ guarantees a successful move in the requested direction. A choice of $p \approx 0.93$ moves the learner in the requested direction with probability 0.5 and in each of the other three directions with probability ≈ 0.17.

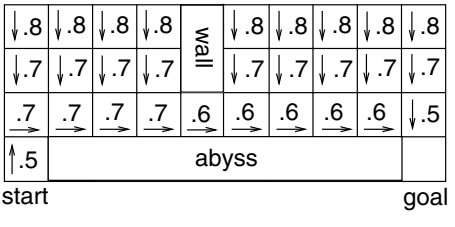


Fig. 2. The grid world map and optimal policy

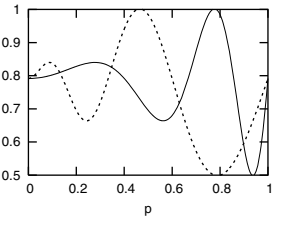


Fig. 3. The two extreme $F(p)$ functions

Our experiments use two different reward functions to guide the learner to the goal, an "informative" one with -1 assigned on every nonterminal transition, -100 on step-

[1] The exact functional form is $F(x, y, p) = \frac{1}{3}w(p, x, y)\sin(4\pi w(x, y, p)) + \frac{19}{24}$, where $w(\cdot)$ is a warping function that applies a different monotonic transformation of the p range for each cell. As a result, the optimum p value is different for every cell. As the cells are traversed in row-major order, the F curve gradually transforms from one extreme in Figure 3 to the other.

ping off the cliff, and $+100$ on reaching the goal cell; and another, "uninformative" reward function with zero reward assigned on all transitions except the one to the goal cell $(+100)$. Because of the discount parameter $\gamma = 0.99 < 1$, the optimal policy is the same under both functions.

The learner initially has no information regarding t, r, or any of the F's. Thus, the challenge is to identify, in each cell, the right direction of travel and the p value that maximizes the probability of this move. We use tile coding in the p variable to approximate the value function for every distinct setting of (x, y, d). Every (x, y, d) triple enjoys a dedicated set of tiles, so there is no generalization across cell boundaries or directions of travel.

3 Initial Empirical Results

This section presents empirical results in three scenarios illustrating the effects of the breadth of generalization on performance. The settings of generalization breadth compared are 1, 3, and 6 tilings, all reasonable choices given the target function curves in Figure 3. The resolution was fixed at 0.04, corresponding to 26, 10, and 6 tiles per tiling in the 1, 3, and 6 tiling cases, respectively.

All experiments in this paper used Q-learning with ϵ-greedy action selection. The parameter settings were: $\alpha = 0.1, \gamma = 0.99$, and $\epsilon = 0.05$, except where indicated otherwise. The Q-value estimates were initialized to 0 on all runs. The metric in all experiments was the value of the start state under the best policy discovered so far (as a percentage of optimal), as determined by an external policy-evaluation module. This model-based evaluation module (value iteration) was unrelated to the model-free algorithm used to learn the policies. Every performance curve in the graphs represents the point-wise average of at least 100 independent runs with all identical settings.

We categorize our empirical findings in three groups:

Experiment A: Initial performance boost due to generalization (regular α). Figure 4 plots early performance obtained using the uninformative reward function (*a*) and the informative one (*b*). The step size α is 0.1, a typical value. The graphs show a performance boost due to generalization when the informative reward function is used, but no observable differences with the uninformative reward function.

Experiment B: Initial performance boost due to generalization (small α). Figure 5 plots performance over the first 50000 episodes, a substantial allotment of learning time. The reward function used is the uninformative one, chosen to control for the apparent advantage enjoyed by broad-generalizing learners in experiment A. (As the analysis in Section 4 will show, the results observed in experiments A and B are due to different causes which this experimental setup serves to isolate.) Decreasing the step size degrades performance for any fixed setting of generalization breadth; however, the extent of this deterioration diminishes as generalization breadth increases. Viewed differently, $\alpha = 0.5$ reveals no observed benefit to generalization. But as α decreases to 0.10 and then to 0.05, generalization becomes increasingly beneficial.

Experiment C: Eventual degradation of performance due to generalization. Experiments A and B demonstrate that generalization can improve performance while the

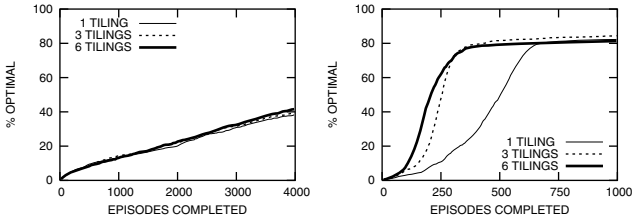


Fig. 4. Early performance with the uninformative (*a*) and informative (*b*) reward functions. The ordering of the curves in *b* is statistically significant at a 0.005 confidence level between episodes 92 and 280

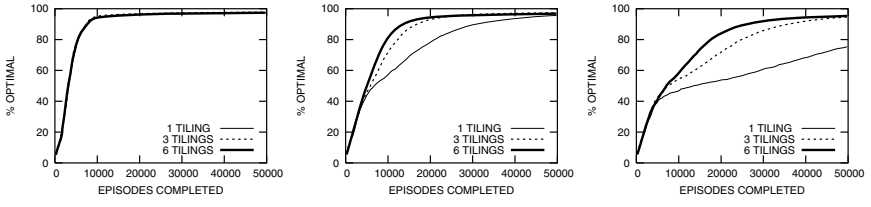


Fig. 5. Performance with the uninformative reward function and three settings of step size: $\alpha = 0.5$ (*a*), $\alpha = 0.1$ (*b*), and $\alpha = 0.05$ (*c*). The ordering of the curves is statistically significant at a 0.005 confidence level between episodes 6100 and 22100 (*b*), and 10000 and 40000 (*c*)

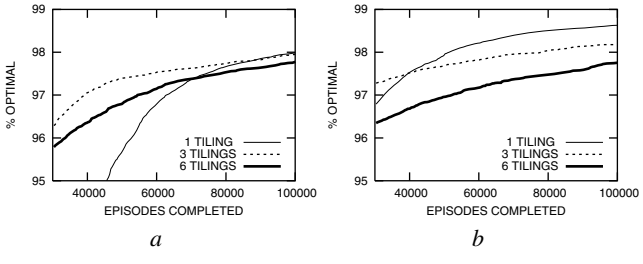


Fig. 6. Long-term performance (episodes 50000–100000) with the uninformative (*a*) and informative (*b*) reward functions. The ordering of the curves in *b* is statistically significant at a 0.005 confidence level staring at episode 55000

policy is undergoing initial development or early refinement. Figure 6, on the other hand, shows that in the final count generalization proves detrimental. Figure 6 was obtained using the uninformative (*a*) and informative (*b*) reward functions. In the former case, the more challenging nature of the task favors the use of generalization for a longer time.

Note that the graphs in Figures 4–6 have vastly different x-axis scales. Moreover, the y-axis scales in Figures 4 and 5 are different from those in Figure 6.

4 Interpretation of Empirical Results

As observed in Section 3, generalization tends to help initial performance but hurts in the long run. This section analyzes the causes of these phenomena. We start by introducing an abstraction of the problem. We categorize state-action pairs as *overestimated*, *underestimated*, and *correctly estimated* with respect to the backup procedure used and the approximate value function $\hat{Q}(s, a)$ for states and actions. In Q-learning, an overestimated state-action pair is characterized by

$$\hat{Q}(s, a) > r(s, a) + \gamma \max_{a' \in \mathcal{A}} \{\hat{Q}(s', a')\},$$

where s' is the successor state. Underestimated and correctly estimated state-action pairs are defined by replacing the "greater than" sign in the above equation with "less than" and "equals" signs, respectively. Note that this terminology is unrelated to the *true* values of state-action pairs under the current policy; state-action pairs are "underestimated," "overestimated," or "correctly estimated" solely with respect to *one-step* updates. Finally, we define a state-action pair (s, a) to be *desirable* if a is a near-optimal action in state s, i.e.,

$$|Q^*(s, a) - \max_{a' \in \mathcal{A}} Q^*(s, a')| < \delta,$$

where Q^* is the optimal value function and $\delta > 0$ is a small constant. *Undesirable* state-action pairs are defined symmetrically.

The effect of generalization on correctly estimated state-action pairs is nonexistent or negligible since backups in such cases generate zero expected error. The effect of generalization on overestimated and underestimated state-action pairs, on the other hand, is significant. In what follows, we analyze these two cases separately. We assume the exploration/exploitation trade-off is addressed using Boltzmann (softmax) action selection, an ϵ-greedy policy, or any other method in which the greedy action $\hat{a}^* = \arg\max_{a \in \mathcal{A}} \hat{Q}(s, a)$ is selected in state s with the greatest probability and the probabilities of selection of the other actions are nondecreasing in their value estimates. We refer to the region to which a value update of a state-action pair (s, a) is generalized as the *vicinity of* (s, a).

4.1 Generalization on Overestimated vs. Underestimated State-Actions Pairs

Generalizing the value update of an overestimated state-action pair (s, a) to nearby state-action pairs will decrease their value estimates and thus reduce the likelihood of selection of the corresponding actions in their respective states. If (s, a) and state-action pairs in its vicinity are undesirable, this generalized update is beneficial *regardless* of whether these state-action pairs are also overestimated. If, on the other hand, some state-action pairs in the vicinity of (s, a) are desirable, generalization is harmful if they are not overestimated. In this latter case, generalization will excessively lower the probability of selection of certain good actions.

Similarly, generalization on an underestimated state-action pair (s, a) is helpful if the state-action pairs in its vicinity are desirable, and may be harmful if there are undesirable pairs with correct or excessive estimates. However, there is an additional benefit

to generalizing on desirable underestimated state-action pairs. Typical domains have continuous value functions. In this case, generalization ensures that the nearby state-action pairs also have favorable estimates *even if they are rarely tried*. Generalization thus accelerates the adoption of better actions in the vicinity of (s, a) as greedy choices, which is increasingly helpful with small step sizes. By contrast, a non-generalizing learner will require more exploratory visits to the vicinity of (s, a) to build up these actions' value estimates.

4.2 Application to the Empirical Results

Generalization improves early performance in experiment A when used with the more informative reward function because the algorithm can more rapidly learn clusters of actions that lead to a fall off the cliff. When such a catastrophic event occurs and a heavy penalty is received (-100), the learner generalizes the outcome to neighboring p values, thus requiring less time to identify directions of travel to avoid *for any value of* p. A non-generalizing learner, on the other hand, needs to visit every p value within resolution to rule out a poor choice of direction. This is an example of the beneficial effects of generalization on overestimated state-action pairs. The uninformative reward function, on the other hand, does not communicate the undesirability of falling off the cliff and leads to no performance improvement with generalization.

The small step sizes in experiment B require a substantial amount of exploratory activity to build up value estimates for better p choices in the vicinity of an already established one—unless generalization across tiles is used, yielding elevated value estimates for those p choices even if they are rarely tried. As a result, generalization improves performance for small step sizes, a benefit of generalization on underestimated state-action pairs. Underestimated state-action pairs are common in experiment B as positive reward propagates from the faraway goal state (the only source of nonzero reward) to the rest of the grid world, one state at a time. Thus, the use of the uninformative reward function ensures that the performance differences are not due to the expedited mastery of cliff avoidance with generalization, as in experiment A.

Once the value estimates become sufficiently accurate (with the optimum actions adopted as greedy choices and the catastrophic actions assigned low values), generalization cannot further improve performance. The negative effects of generalization are, on the other hand, still at work. The empirical results in experiment C confirm that generalization is detrimental at the final stages. As expected, the observed performance degradation is monotonic in the breadth of generalization.

5 Adaptive Generalization

We have confirmed that our empirical findings in Section 3 scale with map size. As an example, Figure 7 shows the early and late performance curves using the informative reward function and a 32×8 grid world. This new map is 6.4 times larger than that of Figure 2 but structurally similar to it.

The empirical results indicate that broad generalization is helpful at the early stages of learning but detrimental in the final count, suggesting that *online adjustment* of gen-

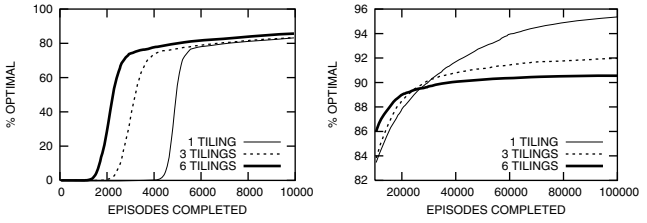


Fig. 7. Performance in a 32×8 grid world: episodes 0–10000 (a) and 10000–100000 (b). The ordering of the curves is statistically significant at a 0.005 confidence level between episodes 2000 and 5600 (a) and starting at episode 40000 (b)

eralization breadth would yield the optimal approach. To this end, we implemented an adaptive algorithm as follows. For every state-action pair (s, a), the method maintains a *reliability index* $\rho(s, a)$ that expresses the learner's confidence in $\hat{Q}(s, a)$, ranging from 0 (unreliable) to 1 (reliable). The reliability indices (initialized to 0) are stored in a tiling organization with the same resolution as the organization for the Q-values themselves. Backups of $\hat{Q}(s, a)$ that yield a large error *lower* the reliability indices for (s, a) and nearby state-action pairs; backups that result in a small error *increase* those reliability indices. When performing a backup, the algorithm selects the largest allowable breadth of generalization such that the state-action space covered has an average reliability index of less that $1/2$. This heuristic encourages broad generalization when the value estimates are rapidly changing and discourages generalization when they are near convergence. Note that no actual conversion from one tiling organization to another is necessary when changing generalization breadth: with an appropriate update scheme, a single flat tiling organization can efficiently *simulate* any number of tilings.

In this framework, one needs to specify only the *range* of minimum and maximum generalization breadth to be used, leaving the parameter adjustment to the algorithm. Observe that the adaptive-generalization method varies generalization as needed based not only on the learning stage (*time-variant* generalization), but also on the state-space region (*space-variant* generalization). This facility is valuable because some regions of the state-action space are visited very frequently and favor an early cutback on generalization; other state-action regions are visited only occasionally and would benefit from generalization for a longer time.

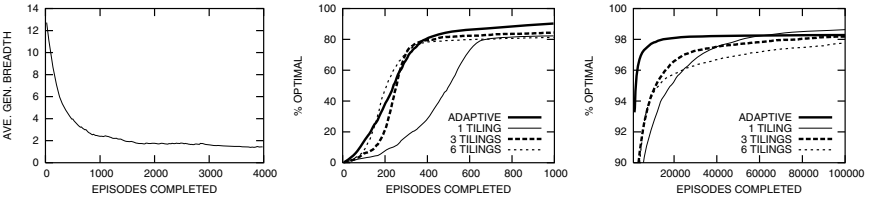
To complete the description of the adaptive-generalization algorithm, it remains to specify how a backup error of a certain magnitude affects the reliability index of the corresponding state-action pair. Various update schemes can be proposed here. Our approach increases the reliability by $1/2$ on zero error and decreases it by $1/2$ on a very large error (50 was an appropriate setting in our domain); the intermediate cases are linear interpolations between these extremes. We generalize each reliability update to its immediate vicinity. In stochastic environments, it may be additionally useful to *decay* the reliabilities periodically. Figure 8 presents the finalized adaptive-generalization algorithm in pseudocode, embedded in Q-learning.

We did not attempt to optimize the above reliability-update rule and used it as an informed first guess. Figure 9*a* illustrates the progress of generalization breadth on a

ADAPTIVE-GENERALIZATION($max\text{-}error$)

Initialize $Q(s,a)$ arbitrarily for all $s \in \mathcal{S}, a \in \mathcal{A}$
$\rho(s,a) \leftarrow 0$ for all $s \in \mathcal{S}, a \in \mathcal{A}$
repeat – current state
  $a \leftarrow \pi(s)$
  Take action a, observe reward r, new state s'
  $error \leftarrow [r + \gamma \max_{a' \in \mathcal{A}} Q(s', a')] - Q(s,a)$
  $num\text{-}tilings \leftarrow \max\{t \geq 1 : \text{avg } \rho(s'', a'') \text{ at most } 1/2\}$
  Update $Q(s,a)$ by $\alpha \cdot error$ using generalization breadth $num\text{-}tilings$
  $\rho(s,a) \leftarrow \left[\rho(s,a) + \left(\frac{1}{2} - \frac{|\,error\,|}{max\text{-}error}\right)\right]_0^1$
  $\pi \leftarrow \epsilon$-greedy w.r.t. Q
until converged

Fig. 8. Adaptive generalization method in pseudocode. The left arrow "←" denotes assignment; $[x]_b^a = \max\{\min\{x, a\}, b\}$ denotes the bounding operation



a *b* *c*

Fig. 9. Adaptive method in the 10×4 grid world: per-episode average generalization breadth, smoothed (*a*); and comparative performance, episodes 0–1000 (*a*) and 1000–100000 (*b*). At a 0.005 confidence level, the adaptive method is superior between episodes 450 and 49000

typical run in this scheme. Figures 9*b* and 9*c* demonstrate that even this "first guess" approach to varying generalization is superior to fixed settings of generalization breadth between episode numbers 450–49000, which arguably covers any reasonable allotment of training time in this domain.

To see why the 1-tiling (no generalization) learner eventually overtakes the adaptive learner, observe that in online RL the learner typically discovers the optimal policy much sooner than its *exact* value function. Indeed, to obtain the optimal policy the learner need only get the relative values of the states right; the actual estimates can be arbitrarily far from the true values. Even after a near-optimal policy has been discovered, the adaptive learner thus continues to see a steady drift of the values as positive reward propagates from the goal state to the rest of the grid world, one state at a time. Faced with this continual change, the above adaptive rule is too slow to cut generalization. This minor drift is easy to detect and correct for with a more informed reliability-update rule. At the same time, even our relatively simple update rule results in good performance. We conclude that even unsophisticated schemes for varying generalization breadth are generally preferable to any fixed setting.

6 Conclusions

This paper explores parameterization issues in tile coding, a widely adopted function-approximation method for reinforcement learning. In particular, we present a precise empirical study of the effect of *generalization breadth* on the performance of a tile-coding approximator. Our findings demonstrate that generalization helps at the early stages of learning but invariably hurts past a certain point. As a result, *no single setting* achieves the best performance throughout the learning curve. We pinpoint the causes of this phenomenon and build on our analysis to propose a novel technique for *automatically adjusting* generalization breadth as needed in different regions of the state-action space (*space-variant* generalization) and at different learning stages (*time-variant* generalization). We experimentally show the superiority of varying the generalization breadth in this way to any fixed parameterization. Our adaptive-generalization method is generic and can be advantageously applied in any setting in which tile coding is used.

Acknowledgments

This research was supported in part by NSF CAREER award IIS-0237699 and an MCD fellowship. The authors are thankful to Lilyana Mihalkova for her feedback on an earlier version of this manuscript.

References

1. Santamaría, J., Sutton, R., Ram, A.: Experiments with reinforcement learning in problems with continuous state and action spaces (1998)
2. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA (1998)
3. Lin, C.S., Kim, H.: CMAC-based adaptive critic self-learning control. In: IEEE Trans. Neural Networks. Volume 2. (1991) 530–533
4. Stone, P., Sutton, R.S.: Scaling reinforcement learning toward RoboCup soccer. In: Proceedings of the Eighteenth International Conference on Machine Learning, Morgan Kaufmann, San Francisco, CA (2001) 537–544
5. Sutton, R.S.: Generalization in reinforcement learning: Successful examples using sparse coarse coding. In Tesauro, G., Touretzky, D., Leen, T., eds.: Advances in Neural Information Processing Systems 8, Cambridge, MA, MIT Press (1996) 1038–1044
6. Tham, C.K.: Modular On-line Function Approximation for Scaling up Reinforcement Learning. PhD thesis, Cambridge University, Cambridge, England (1994)
7. Watkins, C.J.C.H.: Learning from Delayed Rewards. PhD thesis, Cambridge University (1989)

Appendix: Proof of Theorem 1

Proof. (Sketch.) The theorem can be proven by establishing that any function representable with a t-tiling organization is also representable with a single-tiling organization, and vice versa. The former claim is shown by projecting the t-tiling organization

onto the single-tiling organization and weighting the tiles of the single-tiling organization by the sum of the corresponding tile weights of the t-tiling organization. The latter claim is shown by assigning random weights to the leftmost $t - 1$ tiles of the t-tiling organization (one in each of the first $t - 1$ tilings) and weighting the leftmost tile in the remaining tiling such that the sum of the t tile weights equals the weight of the first tile in the single-tiling organization; the latter weighting operation is repeated iteratively, moving at each step one tile to the right in the t-tiling organization.

Creating Better Abstract Operators

Jonathan Teutenberg and Mike Barley

Department of Computer Science,
The University of Auckland,
New Zealand
{jono, barley}@cs.auckland.ac.nz

Abstract. Using abstract operators for least commitment in planning has been shown to potentially reduce the search space by an exponential factor. However a naive application of these operators can result in an unbounded growth in search space for the worst case. In this paper we investigate another important aspect of abstract operators - that of their construction. Similar to their application, naive construction of an abstract operator may leave you with little search space reduction even in the best case, and significant penalties in the worst. We explain what it means to be a good abstract operator and describe a method of creating good abstract operators.

1 Introduction

The concept of an abstract operator is a natural extension of the least commitment principle of partial order planning (POP). It has been shown that the use of abstract operators can potentially reduce the search space by an exponential factor [1]. If used naively they can produce a worst case increase by an exponential or even unbounded factor [2]. In particular, while abstract operators are designed to reduce the cost of dealing with open precondition flaws, the effect on threats is indeterminate. In this paper we consider the representation and construction of abstract operators with the effect of threats in mind. As we shall see, the representation of an operator itself affects the branching factor of the planning search space.

Section 2 briefly outlines where this work fits in the areas of abstracting operators and least commitment planning. Section 3 begins by describing the role of abstract operators in POP planners. Necessary and contingent preconditions of abstract operators are then defined, and the distinction between them made. Next, abstract operators are looked at as a continuation of the abstraction that operators make over actions and a simple example is given. Section 4 considers the construction of an abstract operator. It is shown that the method by which variables are partitioned can greatly affect the utility of an abstract operator in relation to the branching factor of the search space. This section concludes by giving a method creating valid partitionings of variables and a criteria by which to select a good partitioning from these.

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 206–217, 2005.

2 Related Research

Figure 1 shows a brief taxonomy of the related areas of research. The taxonomy spans two major areas of automated planning: those of least commitment planning and hierarchical planning. Under the umbrella of least commitment we have planners that apply the least commitment approach to step ordering (this includes all POP planners), to support or causal links (such as DESCARTES [3], which takes a pure constraint satisfaction approach to planning) and to operator selection (such as FABIAN [1]). The application of the least commitment principle to operator selection produces abstract operators, thus forging a link with the area of hierarchical planning.

In classic hierarchical planners, such as ALPINE [4] and SHAPER [5], literals are ranked according to their criticality. The more "critical" a literal, the sooner we should plan for it. The most critical literals are planned for first and the least critical literals are planned for last. Intuitively, criticality corresponds to importance. The importance of a literal is measured, to some extent, by how easy it is to achieve that literal. If it is trivial to achieve a literal then it is not considered very critical or important. If it is very difficult to achieve then it is considered very critical or important. These criticality rankings are used to construct levels of abstraction. On a given level of abstraction, all the literals of lower criticality are removed. This creates abstract operators because the operator description at a given level of abstraction does not retain the less critical literals. At a given level of abstraction, if two or more operator schemas only differ in their less critical literals then they are fused into a single operator schema at that abstraction level.

In Hierarchical Task Network (HTN) planners (NOAH [6], NONLIN [7], SHOP2 [8]), instead of trying to achieve goals, they try to perform tasks. Tasks are either primitive (i.e., they can be performed by a single action) or non-primitive. HTN planners use methods to break non-primitive tasks into subtasks. Methods are HTN equivalents of operator schemas. Methods are similar to abstract in that they represent an abstraction actions. However, unlike least-commitment abstract operators and classic hierarchical abstract actions they represent networks of actions.

In this paper we are focusing on the "least commitment" abstract operators that were initially introduced in the FABIAN planner. Least commitment abstract

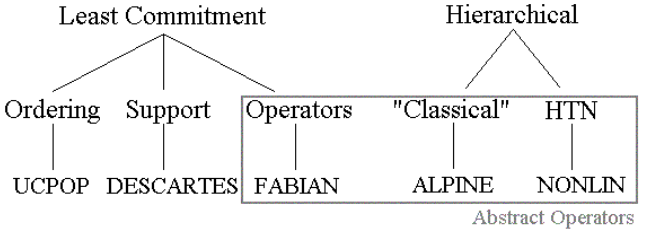


Fig. 1. A taxonomy of related research

operators do not represent suppression of an operator's less critical literals, nor do they represent decomposition of tasks into subtasks. Instead they form a disjunction of operators, any one of which is guaranteed to be applicable if used in place of the abstract operator in a plan. The label "least commitment" has been applied to these abstract operators as they represent a decision (choice between applicable operators) that does not need to be taken immediately. It is entirely possible for a resulting plan to include abstract operators as well as concrete actions, where each refinement of an abstract operator to a concrete action gives a valid solution plan. The following section elaborates on this concept.

3 Abstract Operators

A standard POP algorithm (such as UCPOP [9]) solves planning problems through a simple decision cycle: (1) pick an open condition; (2) support this condition with a causal link (this link can be from either an existing or a new step); and (3) resolve all threats. When a new step is chosen in part (2), the algorithm makes a non-deterministic choice of a single applicable operator. If this decision later is found to be inconsistent with the rest of the plan, the POP algorithm backtracks over this decision trying all possible choices of operator.

An abstract operator takes this operator selection decision and collapses it to a single deterministic point. Rather than choose any one of the applicable operators, a single abstract operator is chosen that represents the set of all these possible operators. As the plan becomes more constrained this set is refined to remove inconsistent choices of operator.

3.1 Effects and Preconditions

Just as standard operators, abstract operators have a set of effects and a set of preconditions. In an abstract operator these effects and preconditions summarise those of its component operators and can be defined as either necessary or contingent.

Definition 1 (Necessary Effects). *A necessary effect is an effect that is in the intersection of the effects of the component operators.*

Definition 2 (Contingent Effects). *A contingent threat is an effect that is in the union of effects of the component operators, but not in the intersection.*

Similar definitions hold for preconditions.

While these definitions of effects and preconditions are quite straightforward there are a number of difficulties involved. For example, what it does it actually mean for an effect to be in the intersection of the effects of its component operators? As we look in more detail at abstract operators these difficulties shall come to light and be dealt with.

3.2 Abstract Operators as Name Variablisation

In planning, an action is defined as a 3-tuple of operator name, effects and preconditions. For example, in the BlocksWorld [10] domain an action to move a block a from a block b to a block c is given by

Name: *move*
Preconditions: $on(a, b), clear(b), clear(c)$
Effects: $on(a, c), clear(b), \neg on(a, b), \neg clear(c)$

Often operators are used to represent sets of these actions. Note that sometimes operators are referred to as operator schema, and actions as operators. An example of an operator for moving a block a from block b to any other block is given by

Name: *move*
Preconditions: $on(a, b), clear(b), clear(X)$
Effects: $on(a, X), clear(b), \neg on(a, b), \neg clear(X)$

This operator is a lifted[11] version of the previous one where X represents the set of all blocks in the domain i.e. $X = \{a, b, c, ...\}$. Thus the operator actually represents the set of actions given by each substitution of a literal block for X.

In this context abstract operators allow a variablisation of the name of an action in addition to variablising literals in the effects and preconditions. Before we show an example of this we must describe the remaining two operators of the BlocksWorld domain. These are the *stack* operator which makes a new stack of blocks by moving a block to the table from another block, and the *unstack* operator which removes a stack of blocks by moving a block from the table onto another block. Completely uninstantiated versions of all three operators are shown below

Name: *move*
Preconditions: $on(X, Y), clear(Y), clear(Z)$
Effects: $on(X, Z), clear(Y), \neg on(X, Y), \neg clear(Z)$

Name: *stack*
Preconditions: $on(X, Y), clear(X)$
Effects: $on(X, table), clear(Y), \neg on(X, Y)$

Name: *unstack*
Preconditions: $on(X, table), clear(X), clear(Y)$
Effects: $on(X, Y), \neg clear(Y), \neg on(X, table)$

Example 1 (Abstract operator for achieving clear(a)). First we take those operators with $clear(a)$ as an effect. The *stack* and *move* operators fit this description so these two are combined to make an abstract operator. In the standard operator notation we have

Name: *N*
Preconditions: $\mathbf{on(X, a)}, \mathbf{clear(X)}, clear(Y)$
Effects: $\mathbf{on(X, Y)}, \mathbf{clear(a)}, \mathbf{\neg on(X, a)}, \neg clear(Y)$

Here N represents the set $\{move, stack\}$. The variable Y can be any block or the *table*. Those effects and preconditions in bold are necessary and those not in bold are contingent. This is just one of many possible abstract operators for achieving $clear(a)$. For example it is possible that one does not wish for the variable Y to represent both the *table* and blocks, and instead have separate contingent effects $on(X, Y)$ and $on(X, table)$. These are important aspects in the construction of abstract operators and will be discussed further in the remaining sections.

4 Partitioning Variables

Let us return to Example 1 and step through the construction of this abstract operator. The first step is to find all operators that are able to achieve the desired precondition $clear(a)$. We have two possible operators:

Name: *move*
  Preconditions: $on(X, a), clear(X), clear(Y)$
  Effects: $on(X, Y), clear(a), \neg on(X, a), \neg clear(Y)$
Name: *stack*
  Preconditions: $on(Z, a), clear(Z)$
  Effects: $on(Z, table), clear(a), \neg on(Z, a)$

If these were combined directly to create an abstract operator we would have

Name: N
  Preconditions: $on(X, a), on(Z, a), clear(X), clear(Z), clear(Y)$
  Effects: $on(X, Y), on(Z, table), \mathbf{clear(a)}, \neg on(X, a), \neg on(Z, a), \neg clear(Y)$

Clearly this is a poor abstraction. We now have an operator with almost twice as many preconditions and effects as the operators it is abstracting. This means there are more open precondition flaws and this operator may cause more threats than would be caused by a better abstraction.

4.1 Unification Through Partitioning

An obvious approach is to combine variables from the component operators so that the intersection of effects or preconditions is greater. In the same way that an abstract operator represents a set of operators, a variable in an abstract operator represents a set of variables from the component operators. In constructing an abstract operator we partition the variables from the component operators into sets that make up the new variables. This is very similar to performing unification on those variables that are members of the same set.

Formally, a *variable partition* is a set of equivalence classes of variables from the component operators of an abstract operator. Each of these equivalence classes is referred to as a *variable equivalence class*. The number of variable equivalence classes of a variable partition P is written $|P|$. The choice of partition is an integral part of the construction of abstract operators. Poor partitioning can lead to an inefficient operator, while a better choice can greatly improve planning efficiency.

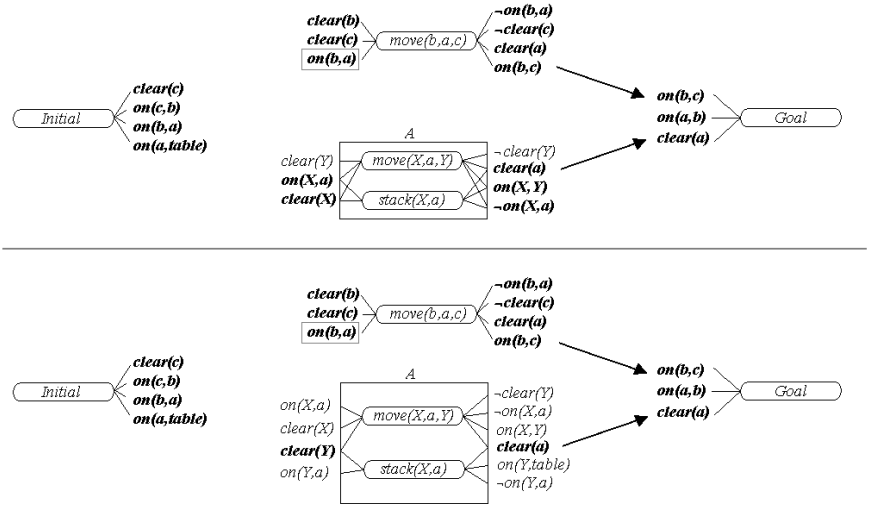


Fig. 2. Two incomplete plans showing an abstract operator with two different variable partitions

To construct the operator given in Example 1 we add the variables X and Z to a single partition, with Y remaining in a second partition. In this case the lteral *table* was included in the same partition as the variable Y, but we shall ignore this fact for now. I must be noted that this is just one possible partitioning of the variables.

Example 2 (An alternative abstract operator). When constructing the abstract operator in Example 1 we could equally have chosen to add Y and Z to the same variable equivalence class and leave X separate. In this case the operator would look like

Name: N
Preconditions: $on(X, a), on(Y, a), clear(X), \mathbf{clear(Y)}$
Effects: $on(X, Y), on(Y, table), \mathbf{clear(a)}, \neg on(X, a), \neg on(Y, a), \neg clear(Y)$

This example's abstract operator is also perfectly valid. So how can we choose between them? One way is to consider their effect on the search space. Figure 2 shows a simple partial plan. An abstract operator A was added to achieve the goal $clear(a)$ with the upper diagram using the partition from Example 1 and the lower diagram using the partition from Example 2. The planner then chose to achieve the goal $on(b, c)$ using a new step. Only a move operator is applicable here. This new step is labelled $move(b, a, c)$ in the diagram. Finally an open precondition of $move(b, a, c)$ was chosen, indicated by a surrounding box, which has been unified with the initial predicate that will provide its causal link. Up to this point both abstract operators appear equal. However, once the new causal link is added (not shown in the diagram) the operator from Example 1

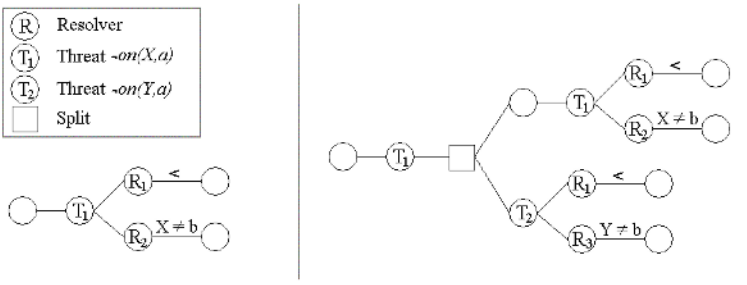


Fig. 3. Branching factor over an open precondition decision point with two different variable partitions

will cause a single non-contingent threat (a threat from a necessary effect), but the operator from Example 2 will cause two contingent threats. Dealing with contingent threats creates additional internal nodes in comparison to necessary effects [2]. Furthermore, there are two of these threats instead of just one necessary. Figure 3 shows the extent of the difference. The diagrams show the overall branching factor over the open precondition decision point for the situations shown in Figure 2. The split node in the tree splits the abstract operator into two new operators, one with only those component operators that cause the threat and one with only those operators that do not. This has been shown to be necessary to avoid exponential increases in branching factor in the worst case [2]. The diagram on the left corresponds to the abstract operator of Example 1 and the right on the right corresponds to that of Example 2. This clearly shows that not only are more internal nodes generated because of the need to split the abstract operator when dealing with contingent threats, but also the number of plans generated is twice as large. In fact, the standard POP algorithm would generate the same four plans at this decision point, so using the operator from Example 2 makes no reduction at all!

4.2 Creating Variable Partitions

Our next step is to generate all valid variable partitions for an abstract operator. Once this is done it is a matter of making a comparison between them and selecting the most promising. Other than the basic restriction that each variable is an element of exactly one variable equivalence class, the only constraint is that no two variables from the same component operator may be in the same equivalence class.

Definition 3 (Minimal Partition). *A minimal partition of an abstract operator is any valid variable partition resulting in the minimal number of variable equivalence classes for that operator.*

Since a minimal partition must be valid, the number of equivalence classes in a minimal partitioning is always equal to the number of variables in the component operator with most variables. It should also be noted that any valid

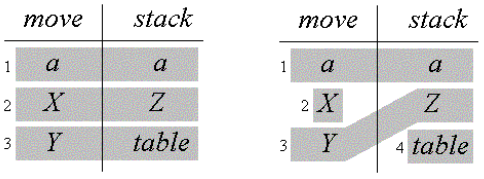


Fig. 4. Two variable partitions for the abstract operator for achieving *clear(a)*

variable partition can be turned into a minimal partition by simply merging equivalence classes. Figure 4 shows the two variable partitions for Example 1(left) and example 2 (right). The grey areas indicate a variable equivalence class of which there are three in the left-hand case and four in the right. Only the left-hand case is considered a minimal partition as for now we are treating the literal *table* as a variable.

Proposition 1. *Take an abstract operator A and a valid non-minimal variable partition P. Then there exists a variable partition P' with $|P'| < |P|$ such that using A with P' will cause a number of threats less than or equal to the number of threats caused by using A with P.*

Proof (of Proposition). Let A be an abstract operator with a non-minimal variable partition P. Then there must exist two variable equivalence classes p_1, p_2 in P s.t merging p_1 and p_2 gives a valid variable partition. Let P' be the variable partition that results from the merging of p_1 and p_2 in P. Let p_3 be the equivalence class in P' that is the union of p_1 and p_2. $|P'| = |P| - 1$ so $|P'| < |P|$ Suppose that A using P' causes some threat with an effect $x(X)$, where $X = p_{x1}, p_{x2}..., p_{xn}$ are the variables in x. Note that the effect $x(X)$ belongs to an abstract operator, so its variables are actually variable equivalence classes.

If $p_3 \notin X$ then $x(X)$ existed in A when using P, so no new threat was made.

If $p_3 \in X$ then in A using P there existed either an effect $x(Y)$, an effect $x(Z)$, or both. Here Y is equal to X with p_3 replaced by p_1 and Z is equal to X with p_3 replaced by p_2. If only one of $x(Y)$ and $x(Z)$ existed in A using P then no new threats were made. If both existed then a threat was removed.

Thus for any non-minimal partition of variables for an abstract operator, there always exists a smaller variable partition that cause the same number or fewer threats.

A simple corollory of Proposition 1 is that any use of variable partition is better than combining no variables (the maximal partition). A further consequence is that we need only consider minimal partitions when constructing abstract operators.

Let the component operators of an abstract operator A be $\{O_1, O_2, ..., O_n\}$, and let the number of variables of a component operator be written $|O_k|$. Let the component operators be ordered such that $|O_i| \leq |O_j| \iff i \leq j$. The exact number of valid minimal variable partitions for A is given by

$$\#partitions = 1 \times_{|O_1|} P_{|O_2|} \times_{|O_1|} P_{|O_3|} \times ... \times_{|O_1|} P_{|O_n|}$$

$$= \frac{|O_1|^{n-1}}{(|O_1| - |O_2|) \times (|O_1| - |O_3|) \times ... \times (|O_1| - |O_n|)} \quad (1)$$

The time taken to produce all valid variable partitions is a constant in the number of partitions given by Equation 1. While this is an expensive operation, it need only be performed once for each possible precondition in a domain. For example, once a variable partition has been chosen for both the $on(X, Y)$ and the $clear(X)$ predicates, no more partitions will need to be computed irrespective of the initial or goal states of a BlocksWorld problem.

4.3 Choosing a Variable Partition

The branching factor, and thus the size, of the search space in planning is determined by the number of open precondition flaws and the number of threat flaws. In addition, backtracking only occurs when a threat is caused that is inconsistant with the constraints of a plan. Abstract operators have been conceived to minimise the effect of branching due to open preconditions, leaving only the effect of threats to worry us. Thus our primary concern is the minimising of threats caused by abstract operators - this is the characteristic by which we must select a variable partition for an abstract operator.

Conjecture 1. Take an abstract operator A with two variable partitions P_1, P_2. Suppose that A using P_2 has fewer effects and preconditions than A using P_1, then, on average, using P_2 causes fewer threats than using P_1.

Conjecture 1 implies that the best way to minimise threats is to minimise the number of effects and preconditions of the operators. An almost equivalent proposal is to maximise the *necessary* preconditions and *necessary* effects of an operator. Both views are consistent with what we have seen in Figure 3.

The difficulty with proving this conjecture is that it concerns the average case. By average we mean the total number of threats occuring in the entire search space over all possible problem instances. There exist partial plans in which abstract operators constructed using the heuristic implied by Conjecture 1 perform worse than some alternatives.

4.4 Invalidating Variable Partitions

A further problem with the selection of a variable partition is that there is no guarantee that two abstract operators resulting from the splitting of an abstract operator will retain a good variable partition. Consider an abstract operator, $AbOp_1$, representing three operators that can achieve a predicate $p_1(X)$:

Name: op_1
Preconditions: $p_2(V), p_4(V)$
Effects: $p_1(X), p_5(R), p_7(R)$

Name: op_2
Preconditions: $p_2(W), p_3(W)$
Effects: $p_1(X), p_5(S), p_6(T)$

Name: op_3
Preconditions: $p_3(Y), p_4(Z)$
Effects: $p_1(X), p_6(U), p_7(Q)$

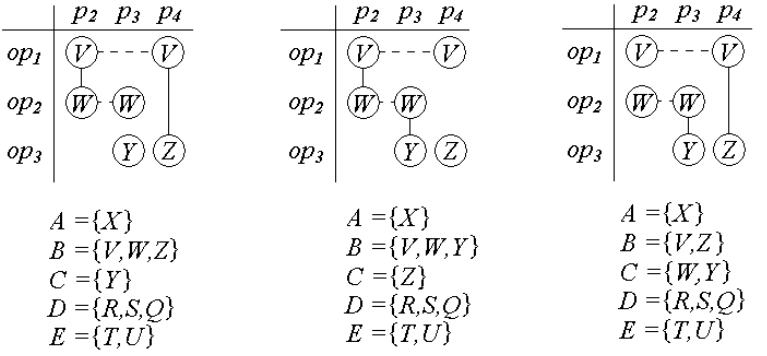


Fig. 5. Three minimal partitionings that produce the minimal number of preconditions

Figure 5 shows the three minimal variable partitions that are considered "best" according to Conjecture 1. The rows refer to each of the three component operators, and the columns to the preconditions. An entry in a table signifies that the precondition exists for the given component operator with the given variable as argument. A dashed line between entries signifies that the same variable is used in both precondition predicates. A solid line between variables signifies that they are both in the same equivalence class, thus each connected component represents a single variable equivalence class. Each connected component that is within a single column results in a single precondition in the resulting abstract operator. It can be easily seen that for all three partitionings the abstract operator has two variables in its four preconditions. Underneath each table its variable partitions are written as sets. Note that such diagrams are appropriate for this example but may not be appropriate in the general case.

The point of interest is the quality of the variable partitions of the operators that result from a split in this abstract operator. For example, suppose that using the left-most variable partition in Figure 5 the effect $p_6(E)$ was found to cause a threat in the plan. The first act of the planner is to split the search space into those plans where $p_6(E)$ is a necessary effect of the abstract operator and those where it is no longer an effect. The later gives rise to plans with the following abstract operator (equivalent to $AbOp_1$ with op_1 removed):

Name: $AbOp_2$
Preconditions: $p_2(B), p_3(B), p_3(C), p_4(B)$
Effects: $\mathbf{p_1}(\mathbf{A}), \mathbf{p_6}(\mathbf{E}), p_7(D)$

However after this split the number of preconditions is no longer minimal. If we had originally used the variable partition in the centre of Figure 5 the abstract operator resulting from the split would have had only three preconditions, of which only two would be contingent.

Of course if the centre variable partition were used instead, it is possible that the planner would encounter a threat caused by the effect $p_7(D)$ which

would result in op_2 being removed and another sub-optimal partition emerging. Similarly for the right-most partition and the effect $p_5(D)$. Thus this is a case in which no partition is able to guarantee the retention of an optimal variable partition throughout the planning process.

Should we not wish to have sub-optimal variable partitions later in planning, there are at least two methods for addressing this problem. It is possible to recompute the variable partition after splitting by searching all possible reassignments of variables in contingent effects and preconditions. However for any reasonable sized abstract operator this is unacceptably expensive computationally. In practice it may be best to precompute the entire forest of abstract operators, with the most general operators as roots, and single operators as leaves.

5 Future Work

Methods of handling literals in operators (such as the occurance of *table* in *stack* and *unstack*) need to be investigated. A simple workaround for partitioning is to wrap each instance of the literal in a pseudo-variable. This has not been proven to give optimal results in all cases, and it is possible that better methods exist.

The case where variables are typed can cause issues during partitioning. Should partitions only contain variables with exactly the same type? Should partitions create a super-type when adding variables with differing domains? How does this impact on the notion of a best partition? All these issues need to be explored.

Another interesting avenue is to move to degrees of contingency. Rather than letting all contingent effects or preconditions be equal, the effect that belongs to one of ten component operators can be seen to be much more contingent than an effect that belongs to nine of ten. This may lead to possible heuristics for ordering threat resolutions during planning, or better partitionings during operator construction.

Finally, empirical evaluations of the different combinations of application, representation and heuristics need to be made.

References

1. Friedman, M., Weld, D.: Least-commitment action selection. In: AIPS Proc. (1996)
2. Teutenberg, J., Barley, M.: Abstract operators: A rather technical report. (2005)
3. Joslin, D.: Passive and Active Decision Postponement in Plan Generation. PhD thesis, University of Pittsburgh (1996)
4. Knoblock, C.: Automatically generating abstractions for problem solving. PhD thesis, Carnegie Mellon University (1991)
5. Fink, E.: Automatic representation changes in problem solving. PhD thesis, Carnegie Mellon University (1999)
6. Sacerdoti, E.: Planning in a hierarchy of abstraction spaces. Artificial Intelligence **5** (1974) 115–135
7. Tate, A.: Generating project networks. In: IJCAI Proc. (1977) 888–893

8. Nau, D., Au, T., Ilghami, O., Kuter, U., Murdock, W., Wu, D., Yaman, F.: Shop2: An htn planning system. Journal of Artifical Intelligence Research **20** (2003) 379–404
9. Weld, D.: An introduction to least commitment planning. AI Magazine **15** (1994) 27–61
10. Winograd, T.: Understanding Natural Language. Academic Press (1972)
11. Robinson, J.A.: A machine-oriented logic based on the resolution principle. Journal of the Association for Computing Machinery **12** (1965) 23–41

A Specialised Binary Constraint for the Stable Marriage Problem*

Chris Unsworth and Patrick Prosser

Department of Computing Science, University of Glasgow, Scotland
{chrisu, pat}@dcs.gla.ac.uk

Abstract. We present a specialised binary constraint for the stable marriage problem. This constraint acts between a pair of integer variables where the domains of those variables represent preferences. Our constraint enforces stability and disallows bigamy. For a stable marriage instance with n men and women we require n^2 of these constraints, and the complexity of enforcing arc-consistency is $O(n^3)$. Although this is non-optimal, empirical evidence suggests that in practical terms our encoding significantly outperforms the optimal encoding given in [7] in both space and time.

1 Introduction

In the Stable Marriage problem (SM) [6, 10] we have n men and n women. Each man ranks the n women into a preference list. So also do the women. The problem is then to produce a matching of men to women such that it is stable. By a matching we mean that there is a bijection from men to women, and by stable we mean that there is no incentive for partners to divorce and elope. A matching is unstable if there are two couples (m_i, w_j) and (m_k, w_l) such that m_i prefers w_l to his current partner w_j, and w_l prefers m_i to her current partner m_k. Stable matching problems occur naturally while matching people to posts [18], such as the allocation of residents to hospitals in the US [17], Canada [5], and Scotland [11]. Variants of the problem occur such as the allocation of groups of students to university accommodation [12], and hard variants have been identified and studied by Irving and Manlove [16]. The problem has also attracted the interest of the constraint programming community [4, 7, 9, 8, 14].

Figure 1 is an instance of the stable marriage problem, and has 6 men and 6 women. Figure 1(a) shows the problem initially, with each man and woman's preference list. Figure 1(b) shows the intersection of the male and female-oriented Gale-Shapley lists (GS-lists) [10], where the GS-lists are reduced preference lists. A man-optimal (woman-pessimal) stable matching can now be found by marrying men (women) to their most (least) preferred choices in there GS-lists.

* The first author is supported by EPSRC. Software support was given by an ILOG SA's academic grant.

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 218–233, 2005.

Men's lists	Women's lists
1: 1 3 6 2 4 5	1: 1 5 6 3 2 4
2: 4 6 1 2 5 3	2: 2 4 6 1 3 5
3: 1 4 5 3 6 2	3: 4 3 6 2 5 1
4: 6 5 3 4 2 1	4: 1 3 5 4 2 6
5: 2 3 1 4 5 6	5: 3 2 6 1 4 5
6: 3 1 2 6 5 4	6: 5 1 3 6 4 2

(a)

Men's lists	Women's lists
1: 1	1: 1
2: 2	2: 2
3: 4	3: 4 6
4: 6 5 3	4: 3
5: 5 6	5: 6 4 5
6: 3 6 5	6: 5 6 4

(b)

Fig. 1. (a) An SM instance with 6 men and 6 women; (b) the corresponding GS-lists

Conversely, we can produce a woman-optimal (man-pessimal) matching by marrying women (men) to their most (least) preferred choice in their GS-lists. An instance of SM admits at least one stable matching and this can be found via the Extended Gale-Shapley algorithm in time $O(n^2)$, where there are n men and n women.

We present a remarkably simple constraint encoding for the stable marriage problem. We introduce a specialised binary constraint with only three methods, where each method is no more than two lines of code. We prove that enforcing arc-consistency in this encoding results in the male-oriented Gale-Shapley lists. We then go on to show how we can extend this encoding by introducing a modest amount of additional code, such that the encoding can be embedded in richer impure problems where the stability of marriages is only part of a larger problem, and the male and female oriented GS-lists are produced. Our empirical results suggest, that although our encodings has $O(n^3)$ time complexity, it significantly outperforms the optimal encoding proposed in [7] in both space and time. In the presentation that follows we will take a one sided, male-oriented, view of the problem. Everything that is presented also has an equivalent and symmetric female-orientation.

2 The Extended Gale-Shapley Algorithm (EGS)

We now describe the male-oriented Extended Gale-Shapley (EGS) algorithm (shown in Figure 2). In particular, we explain what is meant by a *proposal*, an *engagement*, and for a man to become *free*. We will use this later to show that this algorithm and our constraint encoding are equivalent.

The EGS algorithm [10] produces a stable matching between men m_1 to m_n and women w_1 to w_n, where each man (woman) ranks each of the women (men) into preference order. Via a process of proposals from men to women the algorithm delivers reduced preference lists, called GS-lists (Gale-Shapley lists), such that if each man (woman) is paired with his (her) best (worst) partner in their GS-list the marriages will be stable.[1]

[1] Strictly speaking, the given algorithm produces MGS-lists, the male GS-lists. But for the sake of brevity we will refer to them as GS-lists.

```
 1  assign each person to be free
 2  WHILE (some man m is free)
 3  DO BEGIN
 4     w := first woman on m's list
 5     IF (some man p is engaged to w)
 6     THEN assign p to be free
 7     assign m and w to be engaged to each other
 8     FOR (each successor p of m on w's list)
 9     DO BEGIN
10        delete p from w's list
11        delete w from p's list
12        END
13     END
```

Fig. 2. The male-oriented Extended Gale/Shapley algorithm

We will assume that we have an instance I of the stable marriage problem, and that for any person q in I, $PL(q)$ is the ordered list of persons in the original preference list of q and $GS(q)$ is the ordered list of people in the GS-list for q, and initially $GS(q)$ equals $PL(q)$. In a *proposal* from man m to woman w, w will be at the head of the man's GS-list $GS(m)$. This leads to an *engagement* where m is no longer free and all men that w prefers less than m are removed from her GS-list, i.e. the last entry in $GS(w)$ becomes m. Further, when a man p is removed from $GS(w)$ that woman is also removed from his GS-list, i.e. w is removed from $GS(p)$, consequently bigamy is disallowed. Therefore m and w are engaged when m is no longer free, w is head of $GS(m)$, and m is at the tail of $GS(w)$. A man p becomes *free* when p was engaged to w (i.e. the head of $GS(p)$ is w) and w receives a proposal from man m that she prefers to p. On becoming free, p is added to the list of free men and w is removed from $GS(p)$.

The algorithm starts with all men free and placed on a list (line 1). The algorithm then performs a sequence of proposals (lines 2 to 13). A man m is selected from the free list (line 2), and his most preferred woman w is selected (line 4). If w is engaged, then her partner p becomes free. The pair m and w then become engaged (lines 7 to 12).

3 Preliminaries

We assume that each man and woman's preference list has been read into two dimensional integer arrays mpl and wpl respectively. $mpl[i]$ is the preference list for the i^{th} man where $mpl[i][j]$ is the i^{th} man's j^{th} preference, and similarly $wpl[j]$ is the preference list for the j^{th} woman. Using our problem in Figure 1(a), if we consider our third man he will have a preference list $mpl[3] = (1, 4, 5, 3, 6, 2)$.

We also assume we have the inverse of the preference lists, i.e. mPw and wPm, where $mPw[i][j]$ is the i^{th} man's preference for the j^{th} woman and $wPm[j][i]$ is the j^{th} woman's preference for the i^{th} man. Again, considering the third man in Figure 1, his inverse preference list will be $mPw[3] = (1, 6, 4, 2, 3, 5)$,

$mPw[3][2]$ is his preference for the second woman, and that is 6, i.e. woman 2 is in the sixth position of man 3's preference list.[2]

We associate a constrained integer variable with each man and each woman, such that $x[i]$ is a constrained integer variable representing the i^{th} man m_i in stable marriage instance I and has a domain $dom(x[i])$ initially of 1 to n. Similarly, we have an array of constrained integer variables for women, such that $y[j]$ represents the j^{th} woman w_j in I. The values in the domain of a variable correspond to preferences, such that if variable $x[i]$ is assigned the value j this corresponds to m_i being married to his j^{th} choice of woman, and this will be woman $mpl[i][j]$. For example, if $x[2]$ (in Figure 1) is set to 3 then this corresponds to m_2 marrying his third choice, w_1 (and conversely $y[1]$ would then have to be assigned the value 5). Again referring to Figure 1(a) our sixth man's domain is $dom(x[6]) = (1, 2, 3, 4, 5, 6)$, as is everyone else's, and in 1(b) $dom(x[6]) = (1, 4, 5)$. We also assume that we have the following functions, each being of $O(1)$ complexity, that operate over constrained integer variables:

- $getMin(v)$ delivers the smallest value in $dom(v)$.
- $getMax(v)$ delivers the largest value in $dom(v)$.
- $setMax(v, a)$ sets the maximum value in $dom(v)$ to be $\min(getMax(v), a)$.
- $removeValue(v, a)$ removes the value a from $dom(v)$.

We assume that constraints are processed by an arc-consistency algorithm such as AC5 [19] or AC3 [15]. That is, the algorithm has a stack of constraints that are awaiting revision and if a variable loses values then all the constraints that the variable is involved in are added to the stack along with the method that must be applied to those constraints, i.e. the stack contains methods and their arguments. Furthermore, we also assume that a call to a method, with its arguments, is only added to the stack if it is not already on the stack. We'll refer to this stack as the *call stack*.

4 A Binary Stable Marriage Constraint (SM2)

We now give a description of our binary stable marriage constraint, where arc-consistency on such an encoding is equivalent to an application of the male-oriented EGS algorithm. Note that the constraint as described (minimally) cannot be used within a search process, however we will later show how this can be done. Our constraint is binary in that it constrains a man and a woman, such that stability is maintained and bigamy is disallowed. In a stable marriage problem with n men and n women we will then require n^2 of these constraints. We now start by describing the attributes of the constraint, how to construct the constraint, and then the three methods that act upon it. We will use a java-like pseudo-code such that the . (dot) operator is an attribute selector, such that $a.b$ delivers the b attribute of a.

[2] The inverse of the preference lists can be created when reading in the preference lists such that $mPw[i][mpl[i][j]] = j$, and this does not affect the overall complexity of constructing our model.

4.1 The Attributes

A binary stable marriage constraint (SM2) is an object that acts between a man and a woman, and has the following attributes:

- x and y are constrained integer variables representing the man and the woman that are constrained.
- xPy is x's preference for y. That is, if x corresponds to the i^{th} man and y corresponds to the j^{th} woman then $xPy = mPw[i][j]$.
- yPx is y's preference for x. If y corresponds to the j^{th} woman and x to the i^{th} man then $yPx = wPm[j][i]$.

Therefore a constraint between the ith man and jth woman is constructed via a call to the function $SM2(x[i], mPw[i][j], y[j], wPm[j][i])$. This will construct a constraint object c such that $c.x = x[i]$, $c.y = y[j]$, $c.xPy = mPw[i][j]$, and $c.yPx = wPm[j][i]$. To construct our constraint encoding we would then make a call to $SM2$, as shown, with i and j varying from 1 to n creating the n^2 constraints.

4.2 The Propagation Methods

We now describe three methods that achieve male-oriented arc-consistency between a man x and woman y across a constraint c.

deltaMin(c). This method is called across the constraint c between x and y when the lower bound of the domain of x increases. If the lower bound increases such that the lowest value in the domain of x corresponds to that man's preference for woman y (line 2) then that woman need not consider any man she prefers less than man x. Consequently we can remove from her domain all preferences greater than her preference for man x (line 3).

```
1.   deltaMin(c)
2.    IF getMin(c.x) = c.xPy
3.    THEN setMax(c.y,c.yPx)
```

deltaMax(c). We now describe the method that deals with the situation when the upper bound of a woman $c.y$ is reduced. If woman $y's$ least preferred choice is better than her preference for man x (line 2) then man x is no longer in $y's$ preference list. Therefore we remove woman y from man $x's$ preference list (line 3).

```
1.   deltaMax(c)
2.    IF getMax(c.y) < c.yPx
3.    THEN removeValue(c.x,c.xPy)
```

init(c). The *init* method is called when the constraint is created, and is simply a call to *deltaMin*.

```
1.   init(c)
2.    deltaMin(c)
```

5 Comparison to EGS

We now compare the behaviour of our binary constraint model (SM2) to the male-oriented EGS algorithm. In our comparison we will describe steps in the EGS algorithm in *italics* and the SM2 constraint encoding in normal font. For ease of exposition we may refer to a constraint acting between man m and woman w as $C_{x,y}$, where m and w are people in the SM instance, and x and y are the corresponding variables in our constraint encoding. Sometimes we will use m and w as a particular person (rather than m_i and w_j), and x and y as particular variables (rather than $x[i]$ and $y[j]$) for sake of brevity. Additionally, we assume we have the function $fiance(y[j])$ and that it delivers the constrained integer variable $z = x[i]$ where $i = wpl[j][max(dom(y[j])]$, i.e. the least preferred partner of $y[j]$.

- *Initially the EGS algorithm sets all men to be free by adding them to the free list (line 1).* Equivalently, before propagation starts the set of calls $\{init(C_{x[i],y[j]})|1 \leq i, j \leq n\}$ is added to the empty call stack.
- *EGS picks a man m from the free list and he then proposes to his first choice woman w (lines 4 to 7).* Initially, the constraints on the stack will be revised using the *deltaMin* method, called directly via *init*. When executing the call $deltaMin(C_{x,y})$, if y is not $x's$ current favourite (i.e. $min(dom(x)) \neq xPy$) then no action is taken. However, if y is $x's$ favourite the equivalent of a proposal will be made (as described next).
- *When m makes a proposal to w all values that appear in GS(w) after the proposing man are removed (lines 8 to 10), i.e. they become engaged.* With the call $deltaMin(C_{x,y})$, when y is $x's$ favourite, the maximum of $dom(y)$ is set to $y's$ preference for x, therefore removing all less preferred men. Effectively, x and y become engaged.
- *To maintain monogamy EGS removes the newly engaged woman from the GS-lists of all men that have just been removed from her preference list (line 11).* From the action above, the maximum of $dom(y)$ has been lowered, consequently the set of calls $\{deltaMax(C_{x[i],y})|1 \leq i \leq n\}$ are added to the call stack. For a call to $deltaMax(C_{x,y})$, if $y's$ preference for x is greater than the maximum value in $dom(y)$ then $y's$ preference for x has already been removed from $dom(y)$, consequently $x's$ preference for y is removed from $dom(x)$. Therefore, x and y can never be married.
- *In EGS, if m makes a proposal to w, who is already engaged to p, then w's previous fiance p is assigned to be free and added to the free list (lines 5 and 6.)* On initiating the call $deltaMin(C_{x,y})$, $y's$ fiance corresponds to the maximum value in $dom(y)$, because all less preferred men will have been removed (as above). Therefore if y receives a proposal from x via the call $deltaMin(C_{x,y})$, and y prefers x to her current fiance $z = fiance(y)$, the maximum of $dom(y)$ will be set lower than her preference for z and therefore the preference for z will be removed from $dom(y)$. Consequently, the set of calls $\{deltaMax(C_{x[i],y})|1 \leq i \leq n\}$ will be stacked, one of which will be the call $deltaMax(C_{z,y})$, and the preference for y will then be removed from

$dom(z)$. And because y was $z's$ previous favourite the preference for y would have been $min(dom(z))$. Therefore removing that value will increase $z's$ domain minimum, and the set of calls $\{deltaMin(z, y[j])|1 \leq j \leq n\}$ are then added to the stack. And this effectively assigns man z to be free.

6 Proof That GS-Lists Are Produced by the SM2 Encoding

In order to prove that EGS and our SM2 constraint encoding are equivalent we will first show that if EGS removes a value then so does SM2. Conversely, we then prove that if SM2 removes a value then so does EGS. To help make these proofs more readable we will first give some definitions of terms and phrases that will be used.

- $x[i]$ proposes to $y[j]$ when the method $deltaMin(C_{x[i],y[j]})$ is called, where $y[j]$ is $x[i]$'s favourite potential partner, i.e. $j = mpl[i][k]$ where $k = min(dom(x[i]))$.
- $x[i]$ is said to be removed from the domain of $y[j]$ when k is removed from $dom(y[j])$ where $k = wPm[j][i]$. Conversely $y[j]$ is said to be removed from the domain of $x[i]$ when l is removed from $dom(x[i])$ where $l = mPw[i][j]$.
- $x[i]$ is free when there is a call $deltaMin(C_{x[i],y[j]})$ on the call stack, where $y[j]$ is $x[i]$'s favourite potential partner. Therefore, to make $x[i]$ free the set of calls $\{deltaMin(C_{x[i],y[j]})|1 \leq j \leq n\}$ is added to the call stack, although only one of these calls will have an effect, and that is when $y[j]$ is $x[i]$'s most preferred partner.
- $x[i]$ is engaged to $y[j]$ if and only if $x[i]$ is not free and $j = mpl[i][k]$ and $i = wpl[j][l]$, where $k = min(dom(x[i]))$ and $l = max(dom(y[j])$.
- $x[i]$ is rejected by $y[j]$ if k has been removed from $dom(y[j])$, where $k = wPm[j][i]$.

Proof that if EGS removes a person from a GS-list then SM2 will remove the corresponding value from the corresponding variable. We use a proof by cases, considering the situations where EGS can remove people from GS-lists, and the situation where men can become free. There are 3 cases to consider. First we prove that if a proposal in EGS causes people to be removed from GS-lists, then the corresponding proposal in SM2 will result in the same corresponding values being removed from the relevant constrained integer variables. We then prove that if an event in EGS causes a man to be placed on the free list then a similar event in SM2 will add calls to $deltaMin$ to the call stack. Finally we prove that because the order in which the proposals are made does not effect the resulting domains, if EGS removes a person from a GS-list then SM2 will have a corresponding effect.

Lemma 1. In EGS, when a man proposes to a woman and removes values from her GS-list, the constraint encoding will remove the corresponding set of values from the domain of the corresponding variable.

Proof. In EGS a proposal is made from m_i to w_j when m_i is free and w_j is the first entry in $GS(m_i)$, i.e. w_j is m_i's favourite potential partner. All men in $GS(w_j)$ that w_j prefers less than m_i are then removed from $GS(w_j)$, and for each removed man p, w_j is removed from $GS(p)$ thus preventing bigamy.

The proposal is made in SM2 via a call to $deltaMin(C_{x[i],y[j]})$, where $x[i]$ and $y[j]$ correspond to m_i and w_j, and the smallest value in $dom(x[i])$ is $x[i]$'s preference for $y[j]$ (see line 2 of $deltaMin$), i.e. $y[j]$ is $x[i]$'s favourite potential partner. All values greater than $wPm[j][i]$ will be removed from $dom(y[j])$ (line 3 of $deltaMin$). Consequently all values in $dom(y[j])$ corresponding to men she likes less than $x[i]$ will be removed. Since the maximum value in $dom(y[j])$ has decreased the set of calls $\{deltaMax(x[k], w[j])|1 \leq k \leq n\}$ are added to the call stack. When a call to $deltaMax(x[k], y[j])$ is executed and $wPm[j][k]$ is greater than the maximum value in $dom(w[j])$, $mPw[k][j]$ will then be deleted from $dom(x[k])$. □

Lemma 2. If circumstances occur that cause EGS to assign m_i to be free then the same circumstances will cause SM2 to assign $x[i]$ to be free

Proof. The EGS algorithm adds men to the free list under two conditions. The first is when the algorithm is initiated and all men are set to be free. The second is when m_i is rejected by a woman he was previously engaged to, w_j.
When SM2 is initialised the set of calls $\{init(C_{x[i],y[j]})|1 \leq i, j \leq n\}$ will be added to the empty call stack. This will in turn make a call to each method in the set $\{deltaMin(C_{x[i],y[j]})|1 \leq i, j \leq n\}$. This effectively assigns all men to be free.

If man $x[i]$ is rejected by woman $y[j]$, then the value $mPw[i][j]$ will be removed from $dom(x[i])$. Because $x[i]$ previously proposed to $y[j]$, the minimum value in $dom(x[i])$ must have been $mPw[i][j]$. Therefore when $mPw[i][j]$ was removed from $dom(x[i])$, the minimum value in $dom(x[i])$ must have increased and thus caused the set of calls $\{deltaMin(C_{x[i],y[k]})|1 \leq k \leq n\}$ to be put on the call stack, thus effectively assigning $x[i]$ to be free. □

Lemma 3. If EGS removes a person from a GS-list then SM2 will remove the corresponding value from the relevant variable's domain.

Proof. EGS only removes people from a GS-list as a direct result of a proposal (the WHILE loop in Figure 2). From lemma 1 when EGS removes a person from a GS-list SM2 removes the corresponding values from the domain of the relevant variable. From lemma 2, when EGS causes a man to be free so too does SM2. Consequently when EGS removes a person from a GS-list then SM2 will remove the corresponding value from the relevant variable's domain. □

Proof that if SM2 removes a value from a variable then EGS will remove the corresponding person from the relevant GS-List. As above this proof will be split into three parts. First, if in SM2 a value is removed from a variable's domain as a result of a proposal then the same proposal in EGS will

cause the corresponding person to be removed from the relevant GS-List (i.e. the converse of lemma 1). Second, if SM2 assigns a man to be free then EGS in the same circumstances will also assign the same man to be free (the converse of lemma 2). The third combines the previous two to say if SM2 removes a value from the domain of a variable then EGS will remove the corresponding person from the relevant GS-List (similar to lemma 3).

Lemma 4. If in SM2 a value is removed from a variable's domain as a result of a proposal then the same proposal in EGS will cause the corresponding person to be removed from the relevant GS-List.

Proof. In SM2 only two types of domain reductions can occur. The maximum value of $dom(y[j])$ can be set to $wPm[j][i]$ in a call to $deltaMin(C_{x[i],y[j]})$ or $mPw[i][j]$ could be removed from $dom(x[i])$ in a call to $deltaMax(C_{x[i],y[j]})$. Therefore all values removed by SM2 as the result of a proposal must be one of these two types. Because $deltaMin$ only alters the domains of y variables it can only cause calls to $deltaMax$, and likewise $deltaMax$ only removes values from the domains of x variables so can only cause calls to $deltaMin$. And because a call to $deltaMin$ is classed as a proposal the only propagation effect from $deltaMax$ is further proposals.

When a proposal is made in SM2 by a call to $deltaMin(C_{x[i],y[j]})$ The maximum value of $dom(y[j])$ will be set to $wPm[j][i]$, thus removing all men $y[j]$ likes less than $x[i]$. The resulting set of calls $\{deltaMax(C_{x[k],y[j]})|1 \leq k \leq n\}$ will then remove $mPw[k][j]$ from $dom(x[k])$ for all $x[k]$, where $wPm[j][k]$ is greater than the maximum value in $dom(y[j])$, thus removing $y[j]$ from the domains of all men she likes less than $x[i]$.

In EGS when a proposal is made from man m_i to woman w_j all men in $GS(w_j)$ corresponding to men she likes less than m_i are removed. Then w_j is removed from $GS(m_k)$ for all m_k where m_k was removed from $GS(w_j)$. □

Lemma 5. If circumstances occur that cause SM2 to assign man $x[i]$ to be free then the same circumstances will cause EGS to assign the same man m_i to be free.

Proof. In SM2 there are only two events that will cause man $x[i]$ to be placed on the free list. The first is when the set of calls $\{init(C_{x[i],y[j]})|1 \leq i,j \leq n\}$ is placed on the stack. This will in turn call $\{deltaMin(C_{x[i],y[j]})|1 \leq i,j \leq n\}$. This effectively assigns all men to be free.

The other is when the minimum value of $dom(x[i])$ is increased and thus causes the set of calls $\{deltaMin(C_{x[i],y[j]})|1 \leq j \leq n\}$ to be placed on the stack. This can only be due to a call to $deltaMax(C_{x[i],y[j]})$ where $mPw[i][j]$ is the minimum value in $dom(x[i])$. If $mPw[i][j]$ was not the minimum value in $dom(x[i])$ when the call to $deltaMax(C_{x[i],y[j]})$ was made then removing $mPw[i][j]$ from $dom(x[i])$ won't cause any further propagation. If $mPw[i][j]$ is the minimum value in $dom(x[i])$ then $x[i]$ will either be engaged to $y[j]$ or will have not yet proposed and thus be already assigned free. Because $y[j]$ can only be engaged to

one man at a time, however many values are removed from $dom(y[j])$ as a result of a call to $deltaMin(C_{x[k],y[j]})$ (a proposal from some other man x[k]) only one man $x[i]$ can be placed on the free list that was not already there, where $x[i]$ was previously engaged to $y[j]$.

The EGS algorithm adds men to the free list under two conditions. The first is when the algorithm is first started and all men are set to be free. The second is man m_i is placed on the free list if he is rejected by a woman w_j, where m_i and w_j were previously engaged. □

Lemma 6. If SM2 removes a value from a domain then EGS will also remove the corresponding person from the relevant GS-List.

Proof. SM2 only removes values as a direct result of a proposal. From lemma 4 if in SM2 a value is removed from a variable's domain as a result of a proposal then the same proposal in EGS will cause the corresponding person to be removed from the relevant GS-List. From lemma 5 if circumstances occur that cause SM2 to assign $x[i]$ to be free then the same circumstances will cause EGS to assign m_i to be free. Consequently, if SM2 removes a value from a domain then EGS will remove the corresponding person from the relevant GS-List. □

Proof that GS-Lists are produced. This section simply pulls together the previously presented lemmas to prove that the GS-Lists are produced.

Theorem. When a SM instance is made arc consistent using SM2, the resulting domain values will be the equivalent of the GS-Lists produced by EGS from the same SMP.

Proof. From lemma 6 if SM2 removes a value from a domain then EGS will also remove the corresponding person from the relevant GS-List, therefore the values removed by SM2 must be a subset of the values removed by EGS. From lemma 3 if EGS removes a person from a GS-list then SM2 will remove the corresponding value from the relevant variable's domain, therefore the values removed by EGS must be a subset of the values removed by SM2. Therefore the set of values removed by SM2 must be the equivalent of the set of people removed by EGS, and thus the remaining values will be equivalent. Therefore SM2 produces the equivalent of the GS-Lists. □

7 Complexity of the Model

In [10] section 1.2.3 it is shown in the worst case there is at most $n(n-1)+1$ proposals that can be made by the EGS algorithm, and that the complexity is then $O(n^2)$. We argue that the complexity of our SM2 encoding is $O(n^3)$. First we claim that the call to our methods $deltaMin()$ and $deltaMax()$ is of complexity $O(1)$.

When the lower bound of a man $m's$ domain is increased (or is initially given) this will result in the n calls $\{deltaMin(m, woman[i]) | 1 \leq i \leq n\}$. Only one of

these will result in the reduction of a woman w's domain, and this single event will result in the n additional calls $\{deltaMax(man[i], w | 1 \leq i \leq n\}$. This then amounts to $O(n)$ steps. We assume that a lower bound can change at most $n-1$ times, and that this can happen to all n men. Therefore in total we have at most $O(n^3)$ steps performed.

8 Enhancing the Model

The full GS-Lists are the union of the male and female Gale-Shapley lists remaining after executing male and female oriented versions of EGS. It has been proven that the same lists can be produced by running the female orientated version of EGS on the male-oriented GS-lists [10]. Because SM2 produces the same results as EGS the full GS-Lists can be produced in the same way. But because of the structure of this specialised constraint it is also possible to combine the male and female orientated versions of SM2 into one constraint. This combined gender free version of SM2 will then produce the full GS-List with only one run of the arc-consistency algorithm.

The SM2 constraint as presented so far has only considered domain values being removed by the constraint's own methods. If we were to use the constraint to find all possible stable matchings, unless arc consistency reduces all variable domains to a singleton, it will be necessary to assign and remove values from variable domains as part of a search process. Therefore, we need to add code to SM2 to maintain consistency and stability in the event that domain values are removed by methods other than those within SM2. It is important to note that these external domain reductions could also be caused by side constraints as well as a search process.

There are four types of domain reduction that external events could cause: a variable is instantiated; a variable's minimum domain value is increased; a variable's maximum domain value is reduced; one or more values are removed from the interior of a variable's domain. We now describe two new methods, *inst* and *removeValue*, and the enhancements required for *deltaMin*. We note that *deltaMax* does not need to change, and describe the required enhancements for incomplete preference lists.

inst(c). The method $inst(c)$ is called when a x variable is instantiated.

```
1.   inst(c)
2.    IF getValue(c.x) = c.xPy
3.    THEN setValue(c.y,c.yPx)
4.    ELSE IF getValue(c.x) > xPy
5.         THEN setMax(c.y,c.yPx - 1)
6.         ELSE removeValue(c.y,c.yPx)
```

If x prefers to be matched to y (line 2) then y must be instantiated to her preference for x to maintain consistency (line 3). However, if x is matched to someone that he prefers less than y (line 4) then in order to maintain stability y can only marry people that she prefers to x (otherwise x and y will elope).

Consequently we delete all less preferred men, including x, from her domain (line 5). Finally, if x would not prefer to be matched to y than his current partner then we simply deny y the opportunity of marrying x (line 6). Note that $getValue(v)$ delivers the integer value that has been assigned to variable v, $setValue(v, a)$ instantiates v to the integer value a, and these methods are of $O(1)$ complexity.

removeValue(c,a). This method is called when the integer value a is removed from $dom(x)$, and this value is neither the largest nor smallest in $dom(x)$.

```
1.   removeValue(c,a)
2.    IF a = c.xPy
3.    THEN removeValue(c.y,c.yPx)
```

If the value a corresponds to x's preference for y (line 2) then the corresponding value must be removed from $dom(y)$, and that is yPx (line 3), and this must be done to prevent bigamy.

Enhancements to deltaMin(c). Up till now we have assumed that all values removed from the head of $dom(x)$ are as a result of x being rejected by some y variables. We now drop this assumption. We add a new conditional (line 4 below) to address the situation where x would prefer to be matched to y than to his current best preference, consequently y must only consider partners she prefers to x (line 5), and this is done to avoid instability.

```
1.   deltaMin(c)
2.    IF getMin(c.x) = c.xPy
3.    THEN setMax(c.y,c.yPx)
4.    ELSE IF getMin(c.x) > c.xPy
5.         THEN setMax(c.y,c.yPx - 1)
```

No enhancements to deltaMax(c). We now consider the situation where some process, other than a proposal, removes values from the tail of a woman's preference list, i.e. when the maximum value of $dom(y)$ changes. The $deltaMax$ method will be called, and the instance continues to be stable as y can still marry partners. However, we need to prevent bigamy, by removing y from the domains of the corresponding x variables removed from the tail of $dom(y)$, and this is just what $deltaMax$ does. Therefore, no enhancement is required.

Incomplete Lists (SMI). The encoding can also deal with incomplete preference lists, i.e. instances of the stable marriage problems with incomplete lists (SMI). For a SM instance of size n we introduce the value $n+1$. The value $n+1$ must appear in the preference lists $mpl[i]$ and $wpl[j]$ as a *punctuation* mark, such that any people after $n+1$ are considered unacceptable. For example, if we had an instance of size 3 and a preference list $PL(m_i) =$ (3,2) we would construct $mpl[i] = (3, 2, 4, 1)$ and this would result in the inverse $mPw[i] = (4, 2, 1, 3)$. Consequently $x[i]$ would always prefer to be unmatched (assigned the value 4) than to be married to $y[1]$. We now need to modify the *init* method such that it sets the maximum value in $dom(x[i])$ to be $mPw[i][n + 1]$. These modifications will only work in the full implementation (i.e. it requires the above enhancements).

9 Computational Experience

We implemented our encodings using the JSolver toolkit [1], i.e. the Java version of ILOG Solver, and also the Koalog [3] and JChoco [2] toolkits. All three implementations performed similarly, therefore we present only our JSolver results. We implemented four constraint encodings for SM. The first two are those presented in [7], namely the $O(n^4)$ forbidden tuples model (FT) and the optimal $O(n^2)$ boolean encoding (Bool). In the FT model there are n^2 binary table constraints and $2.n$ variables with domains 1 to n. The constraints explicitly list the disallowed tuples that correspond to unstable or bigamous assignments. In the Bool encoding there are $2.n^2$ 0/1 variables, where a variable corresponds to a specific man or woman being matched with a given preference. Implication constraints act between 0/1 variables to simulate the topping and tailing of preference lists. The minimal encoding from section 4 (and referred to as SM-lite) produces the intersection of the male and female GS-lists, but cannot be used in search. The full encoding, referred to as SM2, is the full implementation that can be used in search and allows us to enumerate all solutions in a failure-free manner as in the [7] encodings. Our experiments were run on a Pentium 4 2.8Ghz processor with 512 Mbytes of random access memory, running Microsoft Windows XP Professional and Java2 SDK 1.4.2.6 with an increased heap size of 512 Mbytes.

Our first experiment measures the time taken to generate a model of a given SM instance and make that model arc-consistent, i.e. to produce the GS-lists. Figure 3 shows the average time taken to produce the GS-lists for ten randomly generated instances of size 45 up to 1000. Time is measured in seconds, and an entry − means that an out of memory error occurred. We can see that the SM2-lite and SM2 versions perform much the same, and dominate the other models.

This second experiment measures the time taken to generate a model and find all possible stable matchings. Figure 4 shows the average time taken to find all solutions on the same randomly generated instances used in the first experiment. Again it can be seen that the SM2 model dominates the other models.

Figures 3 and 4 raise the following question, if the Bool encoding is optimal then why is it dominated by the SM2 encoding? The main reason for this is that there is no significant difference in the space required to represent variables with significant differences in domain size, because domains are represented as

	size n										
model	45	100	200	300	400	500	600	700	800	900	1000
FT	8.94	-	-	-	-	-	-	-	-	-	-
Bool	0.25	1.2	4.4	-	-	-	-	-	-	-	-
SM2lite	0.16	0.22	0.45	0.89	1.72	2.79	3.96	5.62	7.48	9.46	11.94
SM2	0.16	0.23	0.5	0.94	1.82	2.95	4.21	5.95	8.02	9.82	12.47

Fig. 3. Average computation times in seconds to produce the GS-lists, from 10 randomly generated stable marriage problems each of size n

	size n										
model	45	100	200	300	400	500	600	700	800	900	1000
FT	9.32	-	-	-	-	-	-	-	-	-	-
Bool	0.36	2.02	6.73	-	-	-	-	-	-	-	-
SM2	0.21	0.47	1.97	5.43	10.13	19.22	27.27	43.04	54.98	77.89	124.68

Fig. 4. Average computation times in seconds to find all solutions to 10 randomly generated stable marriage problems each of size n

	FT	bool	SM2
time	$O(n^4)$	$O(n^2)$	$O(n^3)$
constraints	$O(n^4)$	$O(n^2)$	$O(n^2)$
variables	$O(n)$	$O(n^2)$	$O(n)$

Fig. 5. A summary of the complexities of the three constraint models

	size n									
model	100	200	300	400	500	600	700	800	900	1000
Bool	1.9	6.95	-	-	-	-	-	-	-	-
SM2	0.4	1.2	2.87	5	9.15	14.49	20.86	28.65	38.25	52.64

Fig. 6. Average computation times in seconds to find a sex equal solution to 100 randomly generated stable marriage problems each of size n

intervals when values are consecutive. Considering only the variables, the Bool encoding uses $O(n^2)$ space whereas the SM2 model uses $O(n)$ space. For example, with $n = 1300$ the Bool encoding runs out of memory just by creating the 2.1300^2 variables whereas the SM2 model takes less than 0.25 seconds to generate the required 2600 variables each with a domain of 1 to 1300. As can be seen in Figure 5 theoretically the space complexity of the constraints used by SM2 and Bool are the same. In practise this is not the case as SM2 requires exactly n^2 constraints to solve a problem of size n whereas Bool requires $2n+6n^2$ constraints. Therefore the Bool encoding requires more variables and more constraints, resulting in a prohibitively large model.

We now investigate the unweighted sex equal optimisation problem. In the (NP-Hard) sex equal stable marriage problem (SESMP) [10, 13] both men and women are to be equally well matched. In an unweighted SESMP scores are the same as preferences, therefore we find the matching that minimises the absolute difference between the sum of the men's preferences and the sum of the women's preferences. In the SM2 model the values in the domains of variables are preferences, consequently it is straight forward to model the SESMP. All that is required is to add a search goal to minimise the absolute difference between the sum of all x variables and the sum of all y variables. However it is difficult to model the same problem using the Bool constraints. This is because we now have to introduce $2.n$ additional variables with domains 1 to n and an additional

$O(n^2)$ channelling constraints to set those variables. Figure 6 gives the average run time to find the unweighted sex equal matchings to 100 random problems using the SM2 and Bool models.

As can be seen, again the SM2 encoding solves problems faster and can extend to larger instances. The Bool encoding fails to model problems of size 300 and above, whereas the SM2 encoding can solve problems of size 1000 in less than a minute.

10 Conclusion

We have presented a specialised binary constraint for the stable marriage problem. We have demonstrated that the constraint can be used when stable marriage is just a part of a larger, richer problem. Our experience has shown that the constraint is simple to implement in a constraint programming toolkit, such as JSolver, JChoco, and Koalog. The complexity of the constraint is $O(n^3)$, and does not compare well to the theoretically optimal $O(n^2)$ complexity of the boolean encoding in [7]. However, our constraint is more practical than those in [7], typically being able to solve larger problems faster. For example, we have been able to enumerate all solutions to instance of size 1000 in minutes, whereas in [8] the largest problems investigated were of size 60. It is obvious that our model wastes considerable effort. The arc-consistency algorithm typically adds $n - 1$ redundant calls to the revision stack whenever a change takes place, and it is trivial to detect those redundancies. This suggests that it would be easy to design a space efficient n-ary SM constraint that is of complexity $O(n^2)$.

Acknowledgements

We are grateful to ILOG SA for providing us with the JSolver toolkit via an Academic Grant licence. We would also like to thank our reviewers.

References

1. ILOG JSolver. http://www.ilog.com/products/jsolver/.
2. JChoco constraint programming system. http://choco.sourceforge.net/.
3. Koalog Constraint Solver. http://www.koalog.com/.
4. B. Aldershof and O. Carducci. Refined inequalities for stable marriage. *Constraints*, 4:281–292, 1999.
5. Canadian Resident Matching Service. How the matching algorithm works. Web document available at `http://www.carms.ca/matching/algorith.htm`.
6. D. Gale and L. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.
7. I. Gent, R. Irving, D. Manlove, P. Prosser, and B. Smith. A constraint programming approach to the stable marriage problem. In *CP'01*, pages 225–239, 2001.
8. I. Gent and P. Prosser. An empirical study of the stable marriage problem with ties and incomplete lists. In *ECAI'02*, 2002.

9. M. Green and D. Cohen. Tractability by approximating constraint languages. In *Proceedings of CP '03*, volume 2833 of *Lecture Notes in Computer Science*, pages 392–406. Springer-Verlag, 2003.
10. D. Gusfield and R. W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. The MIT Press, 1989.
11. R. Irving. Matching medical students to pairs of hospitals: a new variation on a well-known theme. In *Proceedings of ESA '98*, volume 1461 of *Lecture Notes in Computer Science*, pages 381–392. Springer-Verlag, 1998.
12. R. Irving and D. Manlove. The stable roommates problem with ties. *Journal of Algorithms*, 43:85–105, 2002.
13. A. Kato. Complexity of the sex-equal stable marriage problem. *Japan Journal of Industrial and Applied Mathematics (JJIAM)*, 10:1–19, 1993.
14. I. Lustig and J. Puget. Program does not equal program: constraint programming and its relationship to mathematical programming. *Interfaces*, 31:29–53, 2001.
15. A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.
16. D. Manlove, R. Irving, K. Iwama, S. Miyazaki, and Y. Morita. Hard variants of stable marriage. *Theoretical Computer Science*, 276:261–279, 2002.
17. N. R. M. Program. About the NRMP. Web document available at `http://www.nrmp.org/about_nrmp/how.html`.
18. A. Roth. The evolution of the labor market for medical interns and residents: a case study in game theory. *Journal of Political Economy*, 92(6):991–1016, 1984.
19. P. van Hentenryck, Y. Deville, and C.-M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.

Compositional Derivation of Symmetries for Constraint Satisfaction

Pascal Van Hentenryck[1], Pierre Flener[2], Justin Pearson[2], and Magnus Ågren[2]

[1] Department of Computer Science,
Brown University, Box 1910, Providence, RI 02912, USA
pvh@cs.brown.edu
[2] Department of Information Technology,
Uppsala University, Box 337, SE - 751 05 Uppsala, Sweden
{pierref, justin, agren}@it.uu.se

Abstract. This paper reconsiders the problems of discovering symmetries in constraint satisfaction problems (CSPs). It proposes a compositional approach which derives symmetries of the applications from primitive constraints. The key insight is the recognition of the special role of global constraints in symmetry detection. Once the symmetries of global constraints are available, it often becomes much easier to derive symmetries compositionally and efficiently. The paper demonstrates the potential of this approach by studying several classes of value and variable symmetries and applying the resulting techniques to two non-trivial applications. The paper also discusses the potential of reformulations and high-level modeling abstractions to strengthen symmetry discovery.

1 Introduction

Many applications in constraint satisfaction exhibit natural symmetries which may significantly increase the difficulty of solving. It is thus not surprising that increased attention has been devoted to symmetry breaking in the last decade.

Recent research has mostly focused on *breaking* symmetries, including general symmetry-breaking schemes (e.g., SBDS [1, 14] and SBDD [8, 10]), their efficient implementations (e.g., [21]), and their specialisations for specific applications (e.g., [3, 20]). There has also been a tendency to abstract some of the techniques from particular applications to classes of CSPs [25] or models [9]. However, this line of research assumes that symmetries are given and ignores the tedious and error-prone task of discovering them.

The detection of symmetries is a research avenue pioneered by Freuder [13] and subsequently investigated by many others. Freuder introduced various forms of value interchangeability and his goal was to *discover* and remove symmetries. Unfortunately, it is not tractable to discover many, apparently simple, classes of symmetries in CSPs arising in practical applications. However, see [5] for results on neighbourhood interchangeability, which is a much weaker form of symmetry than considered in this paper.

This research reconsiders the problem of discovering symmetries from a fundamentally different angle. The key insight is to recognise [24] that global (optimisation)

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 234–247, 2005.

constraints [4, 22, 11] offer significant benefits for deriving symmetries compositionally and efficiently. Global constraints are a fundamental aspect of constraint programming: They capture common combinatorial substructures of practical applications and exploit the substructure semantics to obtain more effective filtering algorithms, linear relaxations, and cooperation schemes between solvers. *The main contribution of this research is to show that, once the symmetries of global constraints are specified, it becomes much simpler to derive the symmetries of an application*. This research can also be seen as shifting the burden of discovering symmetries from users to solver designers who are experts in the underlying combinatorics.

The purpose of this paper is to demonstrate the potential of this research direction. The paper makes the following technical contributions:

1. It considers various classes of symmetries and shows how to derive symmetries compositionally and efficiently, starting from global constraints. They include value and variable symmetries, and symmetries in matrix models.
2. It shows how to apply these results to derive the symmetries of two non-trivial applications: scene allocation and progressive party.
3. It shows how various problem reformulations can improve the accuracy of the derivations and suggests a variety of modeling practices to improve symmetry detection.

These technical results should be viewed as a first (small) step towards a comprehensive automated tool for discovering symmetries. What is particularly interesting however is their ability to handle non-trivial applications already, as well as the various research directions they suggest for modeling languages and reformulation tools.

The rest of the paper is organised as follows. After some preliminaries, the paper shows how constraint and function symmetries can be composed for various forms of interchangeability. The techniques are then illustrated on two applications: scene allocation and the progressive party problem. The next section discusses how problem reformulations improve symmetry detection. Finally, symmetries in matrix models are presented and illustrated.

2 Preliminaries

This section defines the main concepts used in this paper. The definitions are borrowed from [25], which uses them for different purposes. The basic idea is to abstract the set of constraints by a Boolean function which holds if all the constraints are satisfied. Solutions are also represented as functions (assignments), namely from variables to the set of values.

Definition 1. *A* CSP *is a triple $\langle V, D, C\rangle$, where V denotes the set of variables, D denotes the set of possible values for these variables, and $C : (V \rightarrow D) \rightarrow Bool$ is a constraint that specifies which assignments of values to the variables are solutions. A* solution *to a CSP $\mathcal{P} = \langle V, D, C\rangle$ is a function $\sigma : V \rightarrow D$ such that $C(\sigma) = true$. The set of solutions to a CSP $\mathcal{P}$ is denoted by $Sol(\mathcal{P})$.*

Many practical problems involve the optimisation of objective functions and much research in recent years has focused on applying filtering algorithms to prune the resulting "optimisation" constraints (e.g., [23, 11]). In general, in existing languages and systems, these optimisation constraints are expressed using auxiliary variables. However, it is more elegant from a modeling standpoint, and more effective when deriving symmetries, to capture these functions directly.

Definition 2. *A* global function *over variables V and values D is a function $f : (V \rightarrow D) \rightarrow \mathcal{N}$.*

A constraint optimisation problem (COP) consists of minimising an objective function subject to a set of constraints.

Definition 3. *A* COP *is a quadruple $\mathcal{O} = \langle V, D, C, f \rangle$, where $\mathcal{P} = \langle V, D, C \rangle$ is a CSP and f is a global function over V and D. The* optimal value f^* *of $\mathcal{O}$ is the minimal value of f taken by any solution to $\mathcal{P}$, i.e.,*

$$f^* = \min_{\sigma \in Sol(\mathcal{P})} f(\sigma).$$

An optimal solution *of $\mathcal{O}$ is a solution σ of $\mathcal{P}$ whose objective value is optimal, i.e., $f(\sigma) = f^*$. We use $Sol(\mathcal{O})$ to denote $Sol(\mathcal{P})$ in the following.*

The key idea behind this paper is that symmetries can be systematically derived through composition of CSPs (or COPs). The next definition captures compositions of CSPs formally.

Definition 4. *Let $\mathcal{P}_1 = \langle V, D, C_1 \rangle$ and $\mathcal{P}_2 = \langle V, D, C_2 \rangle$ be two CSPs. The* composition *of $\mathcal{P}_1$ and $\mathcal{P}_2$, denoted by $\mathcal{P}_1 \wedge \mathcal{P}_2$, is the CSP $\mathcal{P} = \langle V, D, C_1 \wedge C_2 \rangle$, whose solutions satisfy $Sol(\mathcal{P}) = Sol(\mathcal{P}_1) \cap Sol(\mathcal{P}_2)$.*

3 Value and Variable Interchangeability

There are many applications in resource allocation and scheduling where the exact values taken by the variables are not important. What is significant is which variables take the *same* values or, in other terms, how the variables are clustered. Other applications exhibit weaker notions of value interchangeability, such as the concept of piecewise value interchangeability where only subsets of values are interchangeable. As shown in [25], these symmetries can be broken efficiently during search and it is thus particularly important to discover them automatically.

Definition 5. *Let $\mathcal{P} = \langle V, D, C \rangle$ be a CSP. $\mathcal{P}$ is* value-interchangeable *if, for each solution $\sigma \in Sol(\mathcal{P})$ and each bijection $b : D \rightarrow D$, the function $b \circ \sigma \in Sol(\mathcal{P})$.*

Example 1. Let $V \supseteq \{v_1, v_2, v_3\}$. The CSP $\mathcal{P} = \langle V, D, allDifferent(v_1, v_2, v_3) \rangle$ is value-interchangeable.

We now define piecewise value-interchangeability.

Definition 6. *Let $\mathcal{D} = \{D_1, \ldots, D_n\}$ be a partition of D. A bijection $b : D \to D$ is* piecewise interchangeable *over $\mathcal{D}$ if $\forall v \in D_i : b(v) \in D_i \;\; (1 \leq i \leq n)$.*

Definition 7. *Let $\mathcal{P} = \langle V, D, C\rangle$ be a CSP and $\mathcal{D}$ be a partition of D. $\mathcal{P}$ is* piecewise-value-interchangeable *(PVI) over $\mathcal{D}$ if, for each solution $\sigma \in Sol(\mathcal{P})$ and each piecewise-interchangeable bijection b over $\mathcal{D}$, $b \circ \sigma \in Sol(\mathcal{P})$.*

Note that, if $\mathcal{P} = \langle V, D, C\rangle$ is value-interchangeable, then it is piecewise-value-interchangeable over $\{D\}$. As a consequence, it is easy to compose these two forms of symmetries.

Example 2. Let $V \supseteq \{v_1, v_2, v_3\}$, $D \ni 1$, and consider a constraint *atmost*$(o, d, \langle v_1, \ldots, v_k\rangle)$ which holds for an assignment σ if there are at most o occurrences of d in $\langle \sigma(v_1), \ldots, \sigma(v_k)\rangle$. The CSP $\langle V, D, atmost(2, 1, \langle v_1, v_2, v_3\rangle)\rangle$ is PVI over $\{\{1\}, D \setminus \{1\}\}$.

Value-interchangeability also applies to global functions, in which case the value of a function must not change under various forms of bijection.

Definition 8. *A global function $f : (V \to D) \to \mathcal{N}$ is* value-interchangeable *if, for each assignment $\sigma : V \to D$ and each bijection $b : D \to D$, $f(\sigma) = f(b \circ \sigma)$.*

Example 3. Let $V \supseteq \{v_1, \ldots, v_5\}$ and consider global functions of the form $nbDistinct(v_1, \ldots, v_k)$ which, given an assignment σ, return the number of distinct values in $\langle \sigma(v_1), \ldots, \sigma(v_k)\rangle$. The global function *nbDistinct*$(v_1, \ldots, v_5)$ is value-interchangeable.

Definition 9. *Let $\mathcal{D}$ be a partition of D. A global function $f : (V \to D) \to \mathcal{N}$ is* piecewise-value-interchangeable *over $\mathcal{D}$ if, for each assignment $\sigma : V \to D$ and piecewise-interchangeable bijection b over $\mathcal{D}$, $f(\sigma) = f(b \circ \sigma)$.*

These concepts can be generalised to COPs.

Definition 10. *Let $\mathcal{O} = \langle V, D, C, f\rangle$ be a COP. $\mathcal{O}$ is* value-interchangeable *if, for each solution $\sigma \in Sol(\mathcal{O})$ and each bijection $b : D \to D$, $b \circ \sigma \in Sol(\mathcal{O})$ and $f(\sigma) = f(b \circ \sigma)$.*

Definition 11. *Let $\mathcal{O} = \langle V, D, C, f\rangle$ be a COP and $\mathcal{D}$ be a partition of D. $\mathcal{O}$ is* piecewise-value-interchangeable *over $\mathcal{D}$ if, for each solution $\sigma \in Sol(\mathcal{O})$ and each piecewise-interchangeable bijection b over $\mathcal{D}$, $b \circ \sigma \in Sol(\mathcal{O})$ and $f(\sigma) = f(b \circ \sigma)$.*

In the following, we often assume fixed sets V and D in examples for simplicity and talk directly about the composition and interchangeability of constraints, since they are essentially equivalent to their CSP counterparts.

It is also important to emphasise that all results presented in the next sections have direct counterparts for variable interchangeability. This is due to the fact that the definition of variable interchangeability is essentially similar to value interchangeability. Consider the simplest definition of variable interchangeability.

Definition 12. *Let $\mathcal{P} = \langle V, D, C\rangle$ be a CSP. $\mathcal{P}$ is* variable-interchangeable *if, for each solution $\sigma \in Sol(\mathcal{P})$ and each bijection $b : V \to V$, the function $\sigma \circ b \in Sol(\mathcal{P})$.*

The difference is the composition order of σ and the bijection (which also has a different signature).

4 Composition of Constraint Symmetries

Value symmetries arise in many applications and can be broken efficiently during search. Unfortunately, there is no general efficient algorithm for computing interchangeable values in CSPs [13]. The key insight is that symmetries can be compositionally inferred from global constraints. More precisely, given two constraints (or CSPs) C_1 and C_2, the symmetries of their composition $C_1 \wedge C_2$ can be inferred automatically from the symmetries of C_1 and C_2. The following result is immediate.

Proposition 1. *Let $\mathcal{P}_1 = \langle V, D, C_1 \rangle$ and $\mathcal{P}_2 = \langle V, D, C_2 \rangle$ be two value-interchangeable CSPs. Then, their composition $\mathcal{P}_1 \wedge \mathcal{P}_2$ is value-interchangeable.*

The following example illustrates the result.

Example 4. Let $V \supseteq \{v_1, \ldots, v_6\}$ and let C_1 and C_2 be the constraints *allDifferent* (v_1, v_2, v_3) and *allDifferent*(v_4, v_5, v_6). Then $C_1 \wedge C_2$ is value-interchangeable.

Note that constraints in practice only "constrain" a subset of the variables, although they are formally defined over all variables. The next result specifies how to compose piecewise-value-interchangeable CSPs.

Proposition 2. *Let $\mathcal{P}_1 = \langle V, D, C_1 \rangle$ and $\mathcal{P}_2 = \langle V, D, C_2 \rangle$ be two CSPs. Assume that $\mathcal{P}_i$ is piecewise-value-interchangeable over partition $\mathcal{D}_i$ of D $(1 \leq i \leq 2)$. Then the composition $\mathcal{P}_1 \wedge \mathcal{P}_2$ is piecewise-value-interchangeable over*

$$\mathcal{D} = \{D_1 \cap D_2 \mid D_1 \in \mathcal{D}_1 \;\&\; D_2 \in \mathcal{D}_2 \;\&\; D_1 \cap D_2 \neq \emptyset\}.$$

Proof. First observe that $\mathcal{D}$ is a partition of D. Now let b be a piecewise-interchangeable bijection over $\mathcal{D}$. We show that b is piecewise-interchangeable over $\mathcal{D}_1$. Indeed, consider a set $D_1 \in \mathcal{D}_1$ and a value $d \in D_1$. By definition of $\mathcal{D}$, there exists $D_2 \in \mathcal{D}_2$ such that $I = D_1 \cap D_2$ and $d \in I$. Since b is piecewise-interchangeable over $\mathcal{D}$, $b(d) \in I \subseteq D_1$ and b is piecewise-interchangeable over $\mathcal{D}_1$. Similarly, we can show that b is piecewise-interchangeable over $\mathcal{D}_2$. As a consequence, if $\sigma \in Sol(\mathcal{P}_1 \wedge \mathcal{P}_2)$, then $b \circ \sigma \in Sol(\mathcal{P}_1)$ and $b \circ \sigma \in Sol(\mathcal{P}_2)$. Hence, $b \circ \sigma \in Sol(\mathcal{P}_1 \wedge \mathcal{P}_2)$.

Example 5. Let $D = \{1, \ldots, 10\}$ and let C_1 and C_2 be the constraints *atmost*$(1, 1, \langle v_1, \ldots, v_5 \rangle)$ and *atmost*$(2, 2, \langle v_1, \ldots, v_5 \rangle)$ which are PVI over $\mathcal{D}_1 = \{\{1\}, \{2, \ldots, 10\}\}$ and $\mathcal{D}_2 = \{\{2\}, \{1, 3, \ldots, 10\}\}$ respectively. The composition $C_1 \wedge C_2$ is PVI over

$$\mathcal{D} = \{\{1\}, \{2\}, \{3, \ldots, 10\}\}.$$

It is important to emphasise that the derivation of symmetries using propositions 1 and 2 is polynomial in $|D|$. As a consequence, the compositional symmetry analysis of a CSP is polynomial in $|D|$ and the number of constraints. Of course, it is not guaranteed to be precise, i.e., it may not report all symmetries in the application. However, whenever global constraints are used to model an application, the symmetries appear naturally and the loss of precision is often avoided. Furthermore, we discuss this later in the paper how reformulations may help in addressing this issue.

5 Composition of Function Symmetries

This section shows how to compose function symmetries from global functions. It also shows how to infer symmetries in COPs and how function symmetries can be used to infer symmetries on numerical constraints.

Proposition 3. *Let f_1 and f_2 be two global functions of signature $(V \to D) \to \mathcal{N}$. If f_1 and f_2 are value-interchangeable, then so are $f_1 \star f_2$, where $\star \in \{+, -, \times\}$.*

Of course, the result can be generalised to other operators.

Example 6. Let $V \supseteq \{v_1, \ldots, v_6\}$ and let f_1 and f_2 be the global functions *nbDistinct* (v_1, v_2, v_3) and *nbDistinct*(v_4, v_5, v_6). Then, the global function $3f_1 + 4f_2$ is value-interchangeable.

Proposition 4. *Let $f_1 : (V \to D) \to \mathcal{N}$ and $f_2 : (V \to D) \to \mathcal{N}$ be two global functions. If f_1 and f_2 are piecewise-value-interchangeable over $\mathcal{D}_1$ and $\mathcal{D}_2$ respectively, then $f_1 \star f_2$, where $\star \in \{+, -, \times\}$, is piecewise-value-interchangeable over*

$$\mathcal{D} = \{D_1 \cap D_2 \mid D_1 \in \mathcal{D}_1 \,\&\, D_2 \in \mathcal{D}_2 \,\&\, D_1 \cap D_2 \neq \emptyset\}.$$

We now show how to derive symmetries for COPs by considering both the constraint and the objective function.

Proposition 5. *Let $\mathcal{O} = \langle V, D, C, f\rangle$ be a* COP *and $\mathcal{P} = \langle V, D, C\rangle$. If $\mathcal{P}$ and f are value-interchangeable, then $\mathcal{O}$ is value-interchangeable. If $\mathcal{P}$ is piecewise-value-interchangeable over partition $\mathcal{D}_1$ of D and f is piecewise-value-interchangeable over partition $\mathcal{D}_2$ of D, then $\mathcal{O}$ is piecewise-value-interchangeable over*

$$\mathcal{D} = \{D_1 \cap D_2 \mid D_1 \in \mathcal{D}_1 \,\&\, D_2 \in \mathcal{D}_2 \,\&\, D_1 \cap D_2 \neq \emptyset\}.$$

In many applications, constraints are built from global functions and arithmetic operators. The next proposition shows how to derive symmetries for such constraints.

Proposition 6. *Let $f : (V \to D) \to \mathcal{N}$ be a global function and $\mathcal{D}$ be a partition of D. If f is piecewise-value-interchangeable over $\mathcal{D}$, then the CSP $\langle V, D, f \approx 0\rangle$ is piecewise-value-interchangeable over $\mathcal{D}$ as well, where $\approx \in \{>, \geq, =, \neq, \leq, <\}$.*

6 Scene Allocation

We now illustrate how these results can be used to detect value symmetries on the scene-allocation problem, which consists of producing a movie at minimal cost by deciding when to shoot scenes. Each scene involves a number of actors and at most 5 scenes a day can be filmed. All actors of a scene must be present on the day the scene is shot. The actors have fees representing the amount to be paid per day they spend in the studio. The goal of the application is to minimise the production costs and an optimal solution is an assignment of scenes to days which minimises the production costs. On some

reasonably small instances, a state-of-the-art MIP solver took about 2 minutes and a CP solver took about 8 minutes for solving the problem. By removing value symmetries during search, the execution time of the CP solver fell to below 10 seconds [26].

It is also interesting to quote [25] here: "It should be apparent that the exact days assigned to the scenes have no importance in this application and are fully interchangeable. What is important is how the scenes are clustered together. Our approach does not aim at discovering this fact; rather it focuses on how to exploit it to eliminate the symmetries it induces." *The main contribution of this paper is entirely orthogonal: it shows how the value interchangeability of the scene allocation problem can be automatically derived from the properties of the constraints.*

Consider Figure 1 which depicts an OPL-like model for scene allocation, where the instance data is given in a separate file as typical. The first three lines specify the various ranges for scenes, days, and actors. The next two lines specify the fee of each actor and the set of scenes `S[a]` which actor `a` plays in. The next line specifies the variables and `shoot[s]` represents the day assigned to scene `s`. The constraint `atmost(5,Days,shoot)` is a global cardinality constraint which specifies that at most 5 scenes can be shot every day. The objective function sums the fees of each actor, each actor being paid her fee for each different day in which one of her scenes is shot. Indeed, the expression `all(s in S[a]) shoot[s]` collects the variables associated with the scenes of actor `a` in an array of variables, which is used in the function `nbDistinct`. Observe now that constraint `atmost(5,Days,shoot)` is value-interchangeable. The global function `nbDistinct` is also value-interchangeable (see Example 3). By Proposition 3, the objective function is value-interchangeable. Hence, by Proposition 5, the scene-allocation model in Figure 1 is value-interchangeable. In summary, as mentioned earlier, once the value symmetries of the global objects are known, it is possible to derive value symmetries of the entire model using the results of this paper.

It is also useful to stress the benefits of global constraints. The value symmetries derived on the model above are dramatically more complicated to detect on the MIP model. Indeed, the values are not even explicit in that model, which encodes the scene assignment in terms of 0/1 variables.

```
range Scenes = ...;
range Days = ...;
range Actors = ...;
int fee[Actors] = ...;
{Scenes} S[Actors] = ...;
var Days shoot[Scenes];

minimise
  sum(a in Actors)
    fee[a]*nbDistinct(all(s in S[a]) shoot[s])
subject to
  atmost(5,Days,shoot);
```

Fig. 1. The Scene Allocation Model

7 Progressive Party Problem

The progressive party problem is a traditional benchmark which is often used to compare constraint programming, mathematical programming, and local search. Figure 2 depicts an OPL-like model for this problem, which is a direct translation of the Comet model in [18]. The first three lines specify the ranges, i.e., the boats, the parties, and the periods. The next two lines specify the size of the parties and the capacities of the boats. The variables are declared next and assign a boat `b[g,p]` to party `g` at period `p`. The first set of constraints specifies that a party never visits the same boat twice. The second set of constraints are weighted cardinality constraints which specify that the sizes of the parties visiting a boat during a period cannot exceed the boat capacity. The final set of constraints are again cardinality constraints specifying that two parties meet at most once: a constraint $meetAtmost(\langle v_1,\ldots,v_p\rangle,\langle w_1,\ldots,w_p\rangle,k)$ holds for an assignment σ if $\#\{i \in 1..p \mid \sigma(v_i) = \sigma(w_i)\} \leq k$. Observe that the *allDifferent* constraints are value-interchangeable. The *meetAtmost* constraints are also value-interchangeable. The interesting part in this model are the *weightedAtmost* constraints. A constraint

$$weightedAtmost(\langle s_1,\ldots,s_n\rangle,\langle v_1,\ldots,v_n\rangle,\langle c_1,\ldots,c_m\rangle)$$

holds for an assignment σ if $\forall k \in 1..m : \sum_{i\in S_k} s_i \leq c_k$ where $S_k = \{i \in 1..n \mid \sigma(v_i) = k\}$. This constraint is piecewise-value-interchangeable over $\mathcal{D} = \{D_1,\ldots,D_m\}$, where $D_k = \{i \in 1..m \mid c_i = c_k\}$. As a consequence, our compositional derivation automatically infers that boats with the same capacity are piecewise-value-interchangeable. Note that a similar derivation for the variables infers that parties with the same sizes are piecewise-variable-interchangeable.

```
range Boats = ...;
range Parties = ...;
range Periods = ...;
int size[Parties] = ...;
int cap[Boats] = ...;
var Boats b[Parties,Periods];
solve {
  forall(g in Parties)
    allDifferent(all(p in Periods) b[g,p]);
  forall(p in Periods)
    weightedAtmost(size,
                   all(g in Parties) b[g,p],
                   cap);
  forall(i in Parties, j in Parties: j>i)
    meetAtmost(all(p in Periods) b[i,p],
               all(p in Periods) b[j,p],
               1);
};
```

Fig. 2. The Progressive Party Model

8 Reformulations

The symmetry derivations presented earlier can often be strengthened by model reformulations which can be seen as adaptations to constraint satisfaction of "presolve" techniques used in mixed-integer programming [16]. We present two first reformulations, aggregation and projection.

8.1 Aggregation

The symmetry derivations presented earlier may be suboptimal as the following example indicates.

Example 7. Let $V \supseteq \{v_1, \ldots, v_3\}$ and $D \supseteq \{1, 2\}$. Constraint $C_1 = atmost(2, 1, \langle v_1, v_2, v_3\rangle)$ is PVI over $\mathcal{D} = \{\{1\}, D\backslash\{1\}\}$. Constraint $C_2 = atmost(2, 2, \langle v_1, v_2, v_3\rangle)$ is PVI over $\mathcal{D} = \{\{2\}, D \setminus \{2\}\}$. By Proposition 2, $C_1 \wedge C_2$ is PVI over $\mathcal{D} = \{\{1\}, \{2\}, D \setminus \{1, 2\}\}$. However, $C_1 \wedge C_2$ is also PVI over $\mathcal{D} = \{\{1, 2\}, D \setminus \{1, 2\}\}$, which is stronger.

This precision loss can be remedied by modeling the problem more globally using, say, a global cardinality constraint [23]. Again, the observation is that *global constraints are fundamental tools to derive stronger symmetries.*

Example 8. Consider a global cardinality constraint $atmost(\langle o_1, \ldots, o_k\rangle, \langle d_1, \ldots, d_k\rangle, \langle v_1, \ldots, v_n\rangle)$ which holds for an assignment σ if there exist at most o_i occurrences of d_i in $\langle \sigma(v_1), \ldots, \sigma(v_n)\rangle$ $(1 \leq i \leq k)$. It is PVI over $\{D_1, \ldots, D_k, D \setminus (D_1 \cup \cdots \cup D_k)\}$ where $D_i = \{d_j \mid o_j = o_i \ \& \ 1 \leq j \leq k\}$ $(1 \leq i \leq k)$. For instance, $atmost(\langle 1, 2, 1\rangle, \langle 1, 2, 3\rangle, \langle v_1, \ldots, v_n\rangle)$ is PVI over $\{\{1, 3\}, \{2\}, D \setminus \{1, 2, 3\}\}$ since $D_1 = D_3 = \{1, 3\}$.

Of course, a more global modeling of the problem will likely also lead to better propagation. *As a consequence, automated tools for symmetry detection (and modeling in general) should provide aggregation operators exploiting the semantics of constraints.* They can be specified as follows.

Definition 13. *Let C_1 and C_2 be two constraints of signature $\mathcal{C} = (V \to D) \to Bool$. A* compositional aggregator *is a binary operator $\otimes$ of signature $(\mathcal{C} \times \mathcal{C}) \to \mathcal{C}$ such that $C_1 \otimes C_2$ is a single global constraint equivalent to $C_1 \wedge C_2$.*

Example 9. Let $V \supseteq \{v_1, v_2, v_3\}$, $D \supseteq \{1, 2\}$ and constraints $C_1 = atmost(2, 1, \langle v_1, v_2, v_3\rangle)$ and $C_2 = atmost(2, 2, \langle v_1, v_2, v_3\rangle)$. A compositional aggregator of C_1 and C_2 may return the constraint $atmost(\langle 2, 2\rangle, \langle 1, 2\rangle, \langle v_1, \ldots, v_3\rangle)$.

8.2 Projection

Projections, the second class of reformulations considered in this paper, are important in many applications. On the one hand, they are often useful when a general model (e.g., a round-robin sport-scheduling model) is specialised to a specific problem (e.g., the ACC basketball schedule for the 2004 season) by introducing, among others, some

fixed decisions. On the other hand, they are useful in deriving dynamic symmetries, i.e., symmetries not present in the original problem but arising after a number of variable assignments. The following example illustrates the significance of projections when deriving symmetries.

Example 10. Let $V = \{v_1, \ldots, v_5\}$, let C_1 be the constraint *atmost*$(\langle 3, 2\rangle, \langle 1, 2\rangle, \langle v_1, \ldots, v_5\rangle)$ and C_2 be $v_1 = 1$. The CSP $\langle V, D, C_1 \wedge C_2\rangle$ is derived to be PVI over $\mathcal{D} = \{\{1\}, \{2\}, D \setminus \{1, 2\}\}$ since C_1 is PVI over $\{\{1\}, \{2\}, D \setminus \{1, 2\}\}$ and C_2 is PVI over $\{\{1\}, D \setminus \{1\}\}$, which is as strong as possible. However, consider $V' = V \setminus \{v_1\}$ and the constraint C defined as *atmost*$(\langle 2, 2\rangle, \langle 1, 2\rangle, \langle v_2, \ldots, v_5\rangle)$. The CSP $\langle V', D, C\rangle$ is PVI over $\mathcal{D} = \{\{1, 2\}, D \setminus \{1, 2\}\}$.

This example indicates that more symmetries may be available on subproblems when some variables are projected out. Moreover, since the assignment of values to variables is the fundamental operation of many search procedures, projections are an important tool to derive symmetries dynamically. *As a consequence, symmetry detection tools should ideally include projection operators exploiting the semantics of primitive constraints.*

Definition 14. *Let C be a constraint of signature $\mathcal{C} = (V \to D) \to Bool$, and $V' = V \setminus \{v\}$. A* projection operator *for $\mathcal{C}$ wrt $v = d$ is a function $\uparrow_{v=d}$ of signature $\mathcal{C} \to \mathcal{C}'$, where $\mathcal{C}' = (V' \to D) \to Bool$, satisfying*

$$Sol(\langle V, D, C \wedge v = d\rangle) = Sol(\langle V, D, C \uparrow_{v=d} \wedge v = d\rangle.$$

The key intuition here is that constraint $C \uparrow_{v=d}$ is only expressed in terms of variables in V' and does not add or remove any solution to the original problem.

9 Symmetries in Matrix Models

This section considers the derivation of variable symmetries in matrix models, which have been found useful in a variety of applications involving symmetries. In particular, we show how the techniques presented earlier apply to the detection of column symmetries in matrix models. (The derivation of row symmetries is similar.) Figure 3 presents a specification of the progressive party problem using matrix modeling. It is essentially similar to the model presented earlier but uses matrices and rows of matrices directly in constraints. We now show how to systematically derive column-interchangeability on this model.

Formally, a matrix M of variables can be modelled as a bijection $X \times Y \to V$, where X are the row indices of M, Y its column indices, and V its set of variables. For clarity, we use traditional notations: $M[i, j]$ denotes the variable in row i and in column j, $M[i]$ row i, and $M[*, j]$ column j. We assume that all matrices are defined over row indices X and column indices Y.

Definition 15. *A* matrix-CSP *(MCSP) is a triple $\langle M, D, C\rangle$, where M is a matrix of variables, D denotes the set of values for these variables, and $C : (M \to D) \to Bool$ specifies which assignments of values to the variables are solutions. A* solution *to an*

```
range Boats = ...;
range Parties = ...;
range Periods = ...;
int size[Parties] = ...;
int cap[Boats] = ...;
var Boats b[Parties,Periods];
solve {
  forall(g in Parties)
    allDifferent(b[g]);
  weightedAtmost(size,b,cap);
  forall(i in Parties, j in Parties: j>i)
    meetAtmost(b[i],b[j],1);
};
```

Fig. 3. The Progressive Party Matrix Model

MCSP $\mathcal{P} = \langle M, D, C\rangle$ is a function $\sigma : M \rightarrow D$ such that $C(\sigma) = true$. The set of solutions to $\mathcal{P}$ is denoted by $Sol(\mathcal{P})$.

The next definitions specify column interchangeability, a "global" form of variable interchangeability.

Definition 16. *A* column permutation *for a matrix M is a function $\rho : M \rightarrow M$ such that*

$$M[i, j] = \rho(M)[i, b(j)] \quad (i \in X \ \& \ j \in Y)$$

for some bijection $b : Y \rightarrow Y$.

Definition 17. *An MCSP $\mathcal{P} = \langle M, D, C\rangle$ is* column-interchangeable *if, for each solution $\sigma \in Sol(\mathcal{P})$ and each column permutation $\rho : M \rightarrow M$, the function $\sigma \circ \rho \in Sol(\mathcal{P})$.*

Proposition 7. *Let $\mathcal{P}_1 = \langle M, D, C_1\rangle$ and $\mathcal{P}_2 = \langle M, D, C_2\rangle$ be two column-interchangeable MCSPs. Then, their composition $\mathcal{P}_1 \wedge \mathcal{P}_2$ is column-interchangeable.*

Example 11. Consider the matrix model in Figure 3. The constraints *allDifferent* and *meetAtmost* are column-interchangeable. Indeed, the variable (resp. pair) order is not significant in *allDifferent* (resp. *meetAtmost*) and both are applied on rows of the matrix. The global *weightedAtmost* constraint is column-interchangeable, since it applies the same constraint to all columns. It is an aggregation of

```
forall(p in Periods)
  weightedAtmost(size,b[*,p],cap);
```

which cannot be shown column-interchangeable compositionally.

We conclude this section by generalising the results to piecewise interchangeability.

Definition 18. *Let $\mathcal{Y}$ be a partition over Y. A* piecewise column permutation *over $\mathcal{Y}$ for a matrix M is a function $\rho : M \to M$ such that*

$$M[i,j] = \rho(M)[i,b(j)] \quad (i \in X \ \& \ j \in Y)$$

for some piecewise-interchangeable bijection b over $\mathcal{Y}$.

Definition 19. *Let $\mathcal{Y}$ be a partition over Y. An MCSP $\mathcal{P} = \langle M, D, C\rangle$ is* piecewise-column-interchangeable *over $\mathcal{Y}$ if, for each solution $\sigma \in Sol(\mathcal{P})$ and each piecewise column permutation ρ over $\mathcal{Y}$, the function $\sigma \circ \rho \in Sol(\mathcal{P})$.*

Proposition 8. *Let $\mathcal{P}_1 = \langle M, D, C_1\rangle$ and $\mathcal{P}_2 = \langle M, D, C_2\rangle$ be two piecewise-column-interchangeable MCSPs over $\mathcal{Y}_1$ and $\mathcal{Y}_2$ respectively. Then, their composition $\mathcal{P}_1 \wedge \mathcal{P}_2$ is piecewise-column-interchangeable over*

$$\mathcal{Y} = \{Y_1 \cap Y_2 \ \mid \ Y_1 \in \mathcal{Y}_1 \ \& \ Y_2 \in \mathcal{Y}_2 \ \& \ Y_1 \cap Y_2 \neq \emptyset\}.$$

These results naturally generalise to matrix-COPS.

10 Conclusion

This paper reconsidered the problem of discovering symmetries in constraint satisfaction problems by exploiting one of the fundamental aspects of constraint programming: the ability of global constraints to capture combinatorial substructures. The paper showed that, once the symmetries of global constraints are specified, various classes of symmetries can be derived precisely and efficiently in a compositional fashion. The paper studied value and variable interchangeability, as well as column and row interchangeability in matrix models. It also stressed the benefits of traditional reformulations such as aggregation and projection to strengthen symmetry detection. The potential of this approach was demonstrated on two non-trivial applications.

It is important to stress that symmetries can be discovered *fully automatically* in this way. While a human modeller is usually aware of *some* symmetries in a model, such as full column- or row-interchangeability of a matrix CSP, the discovery of more, if not all, symmetries can be tedious and error-prone, especially for *piecewise* interchangeabilities. The latter usually change from one problem instance to the next, so it is safer and faster to let the system discover symmetries. Another strong motivation for the automatic discovery of symmetries is that dynamic symmetries can be discovered as well, using projection operators, as outlined in Section 8.2.

In practice, an implementation of the ideas in this paper would feature a database with the value and variable interchangeability results of each global constraint and global function. In fact, there is not even a need to consider only global constraints and global functions. For example, the CSP $\mathcal{P} = \langle V, D, v_1 < v_2\rangle$, where $V \supset \{v_1, v_2\}$, is piecewise variable-interchangeable over the partition $\{\{v_1\}, \{v_2\}, V \setminus \{v_1, v_2\}\}$ of V, by application of the variable-interchangeability counterparts of Definition 9 and Propositions 4 and 6. Another example is the piecewise value-interchangeability of constraint C_2 in Example 10. Work in these directions has begun [7]. Such a system

will only be limited by the strength of its reformulation operators, as too weak such operators prevent all the (static or dynamic) symmetries from being discovered or lead to insufficiently strong partitions upon piecewise interchangeabilities.

The work of [24] is closely related to ours: (piecewise) *variable* interchangeabilities are discovered compositionally from the model, using the intrinsic interchangeabilities of the constraints rather than their extensional definitions. It is observed that this method works particularly well in the presence of global constraints, as the sets of a variable partition are then likely to have more than two elements, unlike with binary constraints. Our work extends this work by also considering (piecewise) *value* interchangeabilities, (global) functions, and constraint *optimisation* problems, as well as by making the composition results more precise and by presenting reformulation operators. The notion of symmetrical constraint [19] corresponds to our notion of a constraint with (non-piecewise) *variable* interchangeability. Dynamic symmetries have been considered in the context of neighbourhood interchangeability [6, 17]; further investigations have been within the planning domain [12] and in [15].

It is also interesting to relate this research to the automatic modeling project of [2], which uses compositional refinement to transform abstract specifications into constraint programs. Since these transformations may introduce symmetries, [2] proposes to annotate the refinement rules with the symmetries so that they can be broken subsequently. Our *bottom-up* derivation approach is entirely orthogonal to their *top-down* refinement approach: It could in fact be applied as a first step to deduce properties of models before refinement. Both works also address the need for more automation for non-experts, a feature which is currently lacking in constraint programming when compared to MIP technology.

Acknowledgements

All authors are partly supported by institutional grant IG2001-67 of STINT, the Swedish Foundation for International Cooperation in Research and Higher Education. Pascal Van Hentenryck is partly supported by NSF ITR Award DMI-0121495. We thank the referees for their useful comments, especially for pointing out [24].

References

1. Backofen, R., and Will, S. Excluding symmetries in constraint-based search. In *Proceedings of CP'99*. LNCS 1713:73–87. Springer-Verlag, 1999.
2. Bakewell, A.; Frisch, A.M.; and Miguel, I. Towards automatic modelling of constraint satisfaction problems: A system based on compositional refinement. In *Proceedings of CP'03 Workshop on Modelling and Reformulating Constraint Satisfaction Problems*. Available at `http://www-users.cs.york.ac.uk/~frisch/Reformulation/03/`, 2003.
3. Barnier, N., and Brisset, P. Solving the Kirkman's schoolgirl problem in a few seconds. In *Proceedings of CP'02*. LNCS 2470:477–491. Springer-Verlag, 2002.
4. Beldiceanu, N., and Contejean, E. Introducing global constraints in CHIP. *Journal of Mathematical and Computer Modelling* 20(12):97–123, 1994.
5. Choueiry, B.Y., and Noubir, G. On the computation of local interchangeability in discrete constraint satisfaction problems. In *Proceedings of AAAI'98*, pages 326–333, 1998.

6. Choueiry, B.Y., and Davis, A.M. Dynamic bundling: Less effort for more solutions. In *Proceedings of SARA'02*. LNCS 2371:64–82, Springer-Verlag, 2002.
7. Eriksson, M. Detecting Symmetries in Relational Models of CSPs. Master's thesis, Computing Science, Department of Information Technology, Uppsala University, Sweden, 2005.
8. Fahle, T.; Schamberger, S.; and Sellmann, M. Symmetry breaking. In *Proceedings of CP'01*. LNCS 2293:93–107. Springer-Verlag, 2001.
9. Flener, P.; Frisch, A.M.; Hnich, B.; Kızıltan, Z.; Miguel, I.; Pearson, J.; and Walsh, T. Breaking row and column symmetries in matrix models. In *Proceedings of CP'02*. LNCS 2470:462–476. Springer-Verlag, 2002.
10. Focacci, F., and Milano, M. Global cut framework for removing symmetries. In *Proceedings of CP'01*. LNCS 2293:77–92. Springer-Verlag, 2001.
11. Focacci, F.; Lodi, A.; and Milano, M. Optimization-oriented global constraints. *Constraints* 7(3-4):351–365, 2002.
12. Fox, M., and Long, D. Extending the exploitation of symmetries in planning. In *Proceedings of AIPS'02*, 2002.
13. Freuder, E.C. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of AAAI'91*, pages 227–233, 1991.
14. Gent, I.P., and Smith, B.M. Symmetry breaking during search in constraint programming. In *Proceedings of ECAI'00*, pages 599–603, 2000.
15. Gent, I.P.; McDonald, I.; Miguel, I.; and Smith, B.M. Approaches to conditional symmetry breaking In *Proceedings of SymCon'04*, 2004.
16. Johnson, E.; Nemhauser, G.; and Savelsbergh, M. Progress in linear programming-based algorithms for integer programming: An exposition. *INFORMS Journal on Computing* 12:2–23, 2000.
17. Lal, A., and Choueiry, B.Y. Dynamic detection and exploitation of value symmetries for non-binary finite CSPs. In *Proceedings of SymCon'03*, 2003.
18. Michel, L., and Van Hentenryck, P. A constraint-based architecture for local search. In *Proceedings of OOPSLA'02, ACM SIGPLAN Notices* 37(11):101–110, 2002.
19. Puget, J.-F. On the satisfiability of symmetrical constrained satisfaction problems. In *Proceedings of ISMIS'93*. LNAI 689:350–361. Springer-Verlag, 1993.
20. Puget, J.-F. Symmetry breaking revisited. In *Proceedings of CP'02*. LNCS 2470:446–461. Springer-Verlag, 2002.
21. Puget, J.-F. Symmetry breaking using stabilizers. In *Proceedings of CP'03*. LNCS 2833:585–599. Springer-Verlag, 2003.
22. Régin, J.-C. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of AAAI-94*. 1994.
23. Régin, J.-C. Arc consistency for global cardinality constraints with costs. In *Proceedings of CP'99*. LNCS. Springer-Verlag, 1999.
24. Roy, P., and Pachet, F. Using symmetry of global constraints to speed up the resolution of constraint satisfaction problems. In *Proceedings of the ECAI'98 Workshop on Non-Binary Constraints*, pages 27–33, 1998.
25. Van Hentenryck, P.; Flener, P.; Pearson, J.; and Ågren, M. Tractable symmetry breaking for CSPs with interchangeable values. In *Proceedings of IJCAI'03*, pages 277–282. Morgan Kaufmann, 2003.
26. Van Hentenryck, P. Constraint and integer programming in OPL. *INFORMS Journal on Computing* 14(4):345–372, 2002.

Solving the 24 Puzzle with Instance Dependent Pattern Databases

Ariel Felner[1] and Amir Adler[2]

[1] Dept. of Information Systems Engineering,
Ben-Gurion University of the Negev, Beer-Sheva, 84104, Israel
felner@bgu.ac.il
[2] Dept. of Computer Science, Technion, Haifa, 32000, Israel
adlera@cs.technion.ac.il

Abstract. A *pattern database* (PDB) is a heuristic function in a form of a lookup table which stores the cost of optimal solutions for instances of subproblems. These subproblems are generated by abstracting the entire search space into a smaller space called the pattern space. Traditionally, the entire pattern space is generated and each distinct pattern has an entry in the pattern database. Recently, [10] described a method for reducing pattern database memory requirements by storing only pattern database values for a specific instant of start and goal state thus enabling larger PDBs to be used and achieving speedup in the search. We enhance their method by dynamically growing the pattern database until memory is full, thereby allowing using any size of memory. We also show that memory could be saved by storing hierarchy of PDBs. Experimental results on the large 24 sliding tile puzzle show improvements of up to a factor of 40 over previous benchmark results [8].

1 Introduction

Heuristic search algorithms such as A* and IDA* find optimal solutions to state-space search problems. They visit states in a best-first manner according to the cost function $f(n) = g(n) + h(n)$, where $g(n)$ is the actual distance from the initial state to state n and $h(n)$ is a heuristic function estimating the cost from n to a goal state. If $h(s)$ is "admissible" (i.e., is always a lower bound) then these algorithms are guaranteed to find optimal paths.

The *domain* of a search space is the set of constants used in representing states. A *subproblem* is an *abstraction* of the original problem defined by only considering some of these constants and mapping the rest to a "don't care" symbol. A *pattern* is a state of the subproblem. The abstracted *pattern space* for a given subproblem is a state space containing all the different patterns connected to one another using the same operators that connect states in the original problem. A *pattern database* (PDB) stores the distance of each pattern to the goal pattern. These distances are used as admissible heuristics for states of the original problem by mapping (abstracting) each state to the relevant pattern in the pattern database.

Typically, a pattern database is built in a preprocessing phase by searching backwards, breadth-first, from the goal pattern until the whole abstracted pattern space is

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 248–260, 2005.

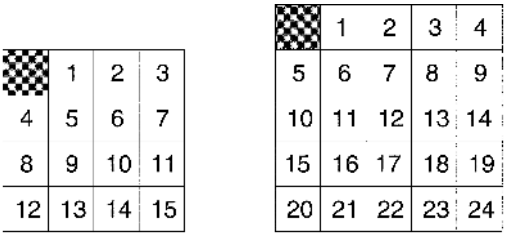


Fig. 1. The 15 and 24 Puzzles in their Goal States

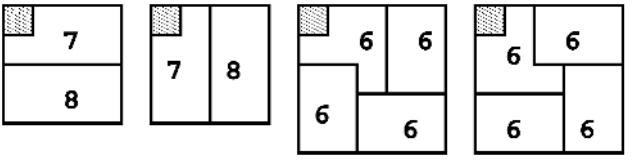


Fig. 2. Partitionings and reflections of the tile puzzles

spanned. Given a state S in the original space, an admissible heuristic value for S, $h(S)$, is computed using a pattern database in two steps. First, S is mapped to a pattern S' by ignoring details in the state description that are not preserved in the subproblem. Then, this pattern is looked up in the PDB and the corresponding distance is returned as the value for $h(S)$. The value stored in the PDB for S' is a lower bound (and thus serves as an admissible heuristic) on the distance of S to the goal state in the original space since the pattern space is an *abstraction* of the original space. PDBs have proven very useful in optimally solving combinatorial puzzles and other problems [1, 7, 8, 3, 6, 2].

The 15 and 24 tile puzzles are common search domains. They consist of 15 (24) numbered tiles in a 4×4 (5×5) square frame, with one empty position - the *blank*. A legal move swaps the *blank* with an adjacent tile. The number of states in these domains is around 10^{13} and 10^{24} respectively. Figure 1 shows these puzzles in their goal configurations.

The best existing optimal solver for the tile puzzles uses *disjoint PDBs* [8]. The tiles are partitioned into disjoint sets (subproblems) and a PDB is built for each set. Each PDB stores the cost of moving *only* the tiles in the given set from any given arrangement to their goal positions and thus values from different disjoint PDBs can be *added* and are still admissible. An $x - y - z$ partitioning is a partition of the tiles into disjoint sets with cardinalities of x, y and z. [8] used a 7-8 partitioning for the 15 puzzle and a 6-6-6-6 partitioning for the 24 puzzle. These partitionings were reflected about the main diagonal (as shown in figure 2) and the maximum between the regular and the reflected PDB was taken as the heuristic.

The speed of search is inversely related to the size of the PDB used, i.e., the number of patterns it contains[5]. Larger PDBs take longer to compute but the main problem is the memory requirements. With a given size of memory only PDBs of up to a fixed

size can be stored. Ordinary PDBs are built such that they are randomly accessed. Thus, storing larger PDB on the disk is impractical and would significantly increase the access time for a random PDB entry unless a sophisticated disk storage mechanism is used. For example a mechanism built on the idea of [11] can be used for storing PDBs on disk. Note, however, that even if the PDBs are stored on disk, disk space is also limited.

A possible solution for this was suggested by [4]. They showed that instead of having a unique PDB entry for each pattern, several adjacent patterns can be mapped to only one entry. In order to preserve admissibility, the compressed entry stores the minimum value among all these entries. They showed that since values in PDBs are locally correlated most of the data is preserved. Thus, we can build large PDBs and compress them into smaller sizes. A significant speedup was achieved using this method for the 15 puzzle and the 4-peg Towers of Hanoi problems.

There are, however, a number limitations to this technique. First, the entire pattern space needs to be generated. Second, only a limited degree of compressing turned out to be effective. For the tile puzzles, it was only beneficial to compress pairs of patterns achieving a memory fold of a factor of two. The largest PDBs that could be built using this technique for the 24 puzzle with one gigabyte of main memory was a $5-5-7-7$ partitioning where the 7-tile PDBs were compressed by a factor of two. This did not gain a speedup over the $6-6-6-6$ partitioning of figure 2 which is probably the best 4-way partitioning of the 24 puzzle.

The motivation for this paper is to use larger PDBs for the 24 puzzle. We want at least an $8-8-8$ partitioning of this domain. A pattern space for 8 tiles has $25 \times 24 \ldots \times 18 = 4.36 \times 10^{10}$ different patterns. Storing three different complete 8-tile PDBs would need 130 gigabytes of memory!!! Using the compressing idea of [4] would not help much and alternative idea should be used.

Another way of achieving reduction in memory requirements is to build a PDB for a specific instance of a start and goal states. Some recent works used this idea for solving the multiple sequence alignment problem, e.g., [9] where the PDB was stored as an Octree. A general formal way for doing this was developed by [10]. They showed that for solving a specific problem instance only a small part of the pattern space needs to be generated. In this paper, we call this idea *Instant dependent pattern databases* (IDPDB). We suggest a number of general enhancements and simplifications to this method and apply them to the 24 puzzle. Experimental results show a reduction of up to a factor of 40 in the number of generated nodes for random instances of this puzzle.

2 Instant Dependent Pattern Databases

We first want to distinguish between the *original search space* where the actual search is performed from the *pattern space* which is a projection of the original search space according to the specification of the patterns (see figure 3). Solving a problem involves two phases. The first phase builds the PDB by performing a breadth-first search backwards from the goal pattern until the entire pattern space is spanned. The second phase performs the actual search in the original search space.

Traditionally, a PDB has a unique entry for each possible pattern. [10] observed that for a given instance of start and goal states only nodes generated by A* (or IDA*)

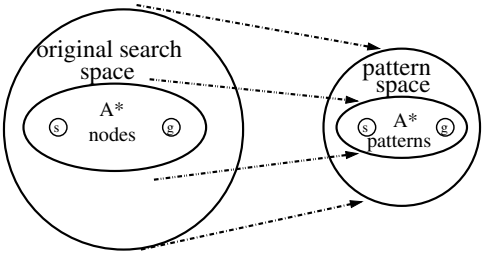


Fig. 3. The projection/abstraction into the pattern space

require a projected pattern entry in the PDB since only these nodes are queried for a heuristic value during the search. In this paper we call these nodes the *A* nodes* and their projections the *A* patterns* (see figure 3). Ideally, we would like to identify and only generate the exact set of the A* patterns but this is impossible. They defined a *focused memory-based heuristic* as a memory based-heuristic (PDB) that is computed only for patterns that are projections of states in the original search space that could be explored by A* in solving the original search problem.

For building the PDB they also search backwards from the goal pattern but are focused on the specific start and goal states. Instead of the usual breadth-first search which searches in all possible directions they activate A* from the goal pattern to the start pattern. In this paper, we call it the *secondary A** in order to distinguish this search from the *primary search* in the original problem which could be performed by any admissible search algorithm (e.g., IDA*). For each pattern expanded by the secondary A*, its g-cost represents the cost of a shortest path from this pattern to the goal pattern in the pattern space and can serve as an admissible h-cost for the original search problem.

After the start pattern is reached by the secondary A* search only a small number of patterns were generated and there is no guarantee that the entire set of the A* patterns was reached. They noted that we can continue to expand nodes after the secondary A* finds an optimal path from the goal pattern to the start pattern to determine optimal g-costs (and thus admissible heuristics) for additional states. We call this the *extended secondary A** phase. We would like to halt the extended secondary A* phase when all the A* patterns are reached but this is a difficult task. They provide a method for identifying a special set of patterns which is a superset of the A* patterns set. We call this set the *ZH set* (after Zhou and Hansen). They halt the extended secondary A* phase after the complete ZH set is generated and are guaranteed that the entire set of the A* patterns is generated.

The definition of the ZH set is as follows. Let U be an upper bound on the cost of the optimal solution to the original problem. The ZH set includes all patterns p_i which have $f(p_i) < U$ in the secondary A* search. It is obvious that all the A* patterns have f-value in the secondary search smaller than U and thus are all included in the ZH set. They also provide a formula for computing and generating the ZH set for a set of disjoint additive PDBs. Let U again be an upper bound on the solution. Let $L = \sum_j (h_j(S))$ be the additive heuristic of the initial state S. Let $\Delta = U - L$. They prove

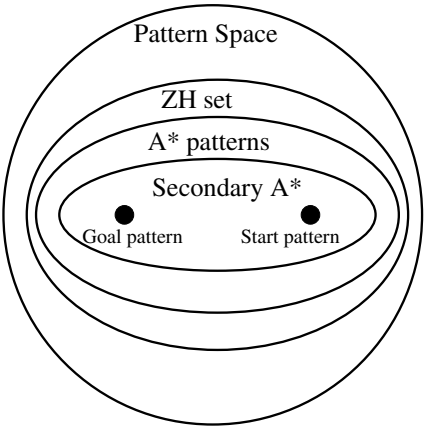


Fig. 4. The different layers of the pattern space

that a disjoint PDB, PDB_j only needs to be calculated for projected patterns p_i having $f(p_i) < h_j(S) + \Delta$. See [10] for more details and proofs.

Figure 4 shows the relations between different sets of patterns. The innermost set includes the patterns generated during the first stage of secondary A* (until the optimal path from the goal pattern to the start pattern is found). The next set includes the A*-patterns, i.e., the patterns queried during the search in the original problem. The next set includes the ZH set which [10] stored in their PDB. The outmost set is the complete pattern space.

They recognized that continuing the extended-A* until the complete ZH set is generated is not always possible due to time/memory limitations. Thus, they introduced their γ factor where $0 < \gamma \leq 1$. They stopped the extended-A* when its f-cost exceeds $\gamma \times U$. With $\gamma < 1$ generating all the A* patterns is not guaranteed. Therefore, when a state on the original space is reached whose projected pattern is not in the PDB due to the γ cutoff, they suggest using a simple quickly computed admissible heuristic instead. They implemented their idea with different values for γ on the multiple-sequence alignment and obtained impressive results.

Note that ordinary PDBs are usually stored in a multi-dimensional array with an entry for each possible pattern. For IDPDB, we need a more sophisticated data structure e.g., a hash table, as only a subset of the patterns is stored.

2.1 Weaknesses of the ZH Method

There are a number of weaknesses in the ZH set approach.

1) Their method needs a fixed amount of memory. Once γ is chosen all the nodes with $f \leq \gamma \times U$ are stored. This is problematic as it is difficult to determine the exact value for γ so that the available memory would be fully used and not be exhausted. Identifying the ZH set and then simplifying it by γ seems not natural and ad hoc.

2) An upper bound, U, for an optimal solution to the original search problem cannot always be found. Furthermore, we need a strict upper bound so as to reduce the ZH set as much as possible.
3) [10] tried out their method only on multiple sequence alignment. The search space of this domain (and also the projected pattern space) has the property that the number of nodes in a given depth d is polynomial in d. This is because the problem is formalized as an n-dimensional lattice with l^n nodes where n is the number of sequence to be aligned and l is the length of the sequences. Even with relatively mediocre U bounds, their ZH set might be significantly smaller than the entire pattern space. This is not true in domains such as the tile puzzles where the number of nodes at depth d is exponential in d. Given any U the ZH set might include the entire domain.

To support these claims experimentally we applied the formula they provide for creating the ZH set for disjoint PDBs on the 15 puzzle. This formula uses an upper bound on the optimal solution. We used the best upper bound possible - the exact optimal solution. We calculated the ZH set with this strict upper bound for a 5-5-5 and a 6-6-3 partitionings of the 15 puzzle on the same 1000 random instances from [8]. The entire 5-5-5 PDB includes 1,572,480 entries, half of them were queried during the search. The average ZH set over the 1000 instances has 1,227,134 pattern - 78% of the entire pattern space. For many difficult instances of this domain, the ZH set actually included the entire pattern space. On those instances the ZH method is useless. Note again that this is when we used a strict upper bound of the actual optimal solution length. For more realistic larger upper bounds the ZH will be even larger. Similar results were obtained for a 6-6-3 partitioning.

3 Dynamically Growing the PDBs

We suggest the following enhancement to Zhou and Hansen's idea. Our enhancement is at least as strong as their method but is simpler to implement and easier to understand. In addition, our idea can fit any size of available memory.

The main point of our idea is to dynamically grow the PDB until main memory is exhausted. Our idea is much more flexible than the method of [10] as it can work with any size of memory and we do not need to decide when to halt the secondary A* extension in advance. Furthermore, we do not need to calculate any upper bounds nor have to build the ZH set. In the preprocessing phase, we continue generating patterns in the extended secondary A* until memory is exhausted. We then start the primary search phase and for each pattern not in the PDB, we use a simple quickly computed admissible heuristic instead.

The following enhancement can better utilize main memory after it was exhausted. There are two data structures in memory. The first is the PDB which is identical to the closed list of the extended secondary A*. The second is the open-list of the extended secondary A*. However, at this point we can remove the open-list from main memory thus freeing a large amount of memory for other purposes such as other PDBs. In fact,

if x is the f-value of the best node in the open list and is also the value of the last node expanded, then all nodes in the open-list with values of x can be added to the PDB before freeing the memory. This is actually expanding these nodes without actually generating their children.

Another way of saving memory is to use IDA* for the secondary search. Here, each new pattern generated is matched against the PDB and if it is missing a new PDB entry is created. However, in the pattern space of eight tiles presented below there are many small cycles since all the other tiles can be treated as blanks. This causes IDA* to be ineffective in this specific pattern space because it cannot prune duplicate nodes due to its depth-first behavior.

3.1 On Demand Pattern Databases

A version of the above idea is called *on-demand pattern database*. Here, we add patterns to the PDB only when they are required during the search. This prevents us from generating large PDBs with patterns that will not be queried.

First, we run the secondary A* from the goal pattern to the start pattern until the start pattern is chosen for expansion. Each pattern expanded by this search is inserted into the PDB. At this point, the preprocessing phase ends and the primary search can begin because the start pattern is already in the PDB. We continue the primary search as long as projected patterns of new nodes are in the PDB (i.e., were expanded by the secondary A* search). When we reach a pattern p not in the PDB and still have free memory, we continue to extend the secondary A* phase until this pattern p is reached and we can return to the primary A* phase.

When memory is exhausted the PDB has reached its final size and the secondary A* is terminated. From this point, each time a heuristic is needed and the relevant pattern is not in the PDB, we consult the quickly computed heuristic.

4 Implementation on the 24 Puzzle

While the above idea is a general one we made some domain dependent enhancements and took special steps to best fit the IDPDB idea to the 24 puzzle.

Generating a PDB consumes time. However, the time overhead of preprocessing the PDBs is traditionally omitted as it is claimed that it can be amortized over the solving of many problem instances. For example, it takes a couple of hours to generate the 7-8 disjoint PDB which was used to solve the 15 puzzle [8]. Yet, the authors ignored this time and only reported the time of the actual search which is a fraction of a second.

We cannot simply omit the time overhead of generating IDPDBs as a new PDB has to be built for each new instance. Therefore, it is irrelevant to apply this idea to small domains such as the 15 puzzle where the running time of the actual search is much smaller than the time overhead of generating the PDB. We cannot see how this method will improve previous running times for such domains. The 24 puzzle is a different story

since it is 10^{11} times larger. Generating the 6-6-6-6 PDB also takes a number of hours. However, a number of weeks were required to solve many of the instances of [8]. Here, the time overhead of generating the PDB can also be omitted when compared to the overall time needed to solve the entire problem.

The above general method is for generating one PDB. When we use disjoint databases, such as the 8-8-8 partitionings for the 24 puzzle (see below), values from the different PDBs are added and therefore three values for each state of the original search space are required. Thus, the on-demand version of IDPDB activates three secondary A* searches in parallel, one for each PDB[1]. Note, that since each move in the tile puzzle domain moves only one tile then at each step we only need to consult the one PDB that includes the tile that has just been moved. Values from the other PDBs can be inherited from the parent and remain identical.

4.1 On Demand Versus Preprocessing

The weakness of the on-demand approach for the 24 puzzle is that three open-lists are maintained at all times but this is wasteful since the open-list can be deleted after memory is exhausted. In the special case of the tile puzzles an open list might have 10 times more nodes than the closed list.

A better way to utilize memory for this domain is to perform the complete secondary A* in the preprocessing phase until memory is exhausted. At this point the closed list which includes all the patterns with valid heuristics is stored in a file on the disk and the entire memory is released. This mechanism is repeated for each PDB until a relevant file with heuristic values is stored on the disk. Memory is better utilized as only one open-list is maintained in memory at any point of time. Furthermore, during the course of the primary search there are no open lists of the secondary searches in main memory. Now, we can load values from the disk files into memory and have a PDB for each of them.

In both the on-demand and the preprocessing variations when an entry was missing from the PDB we took the Manhattan distance (MD) as an alternative simple heuristic for the tiles of the missing entry. In addition, for most variations reported below we also stored the benchmark 6-6-6-6 PDBs (which needed 244 megabytes). We then compared the heuristic obtained from the 8-8-8 PDB to the 6-6-6-6 heuristic and took the maximum between them.

4.2 Improvement 1: Internally Partition the PDB

Note that each of the 8-tile sets of figure 5 can be internally partitioned into a 6-2 partitioning where the 6-tile partition is one of the 6-6-6-6 partitions of figure 2. For example, the 6-2 partitioning is shaded in gray for partition a of figure 5. Instead of taking the MD for the eight tiles of a missing entry, we can use the 6-2 partition of these tiles. For those eight tiles we added the value of the corresponding 6-tile pattern from

[1] Furthermore, this is true for using other combinations of multiple PDBs such as taking the maximum over different PDB values.

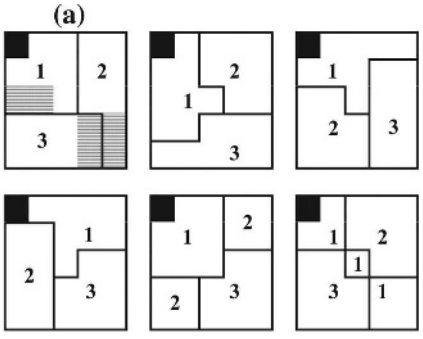


Fig. 5. Different Partitioning to 8-8-8

the 6-6-6-6 PDB to a value of the 2 tiles from a new 2-tile PDB which was also generated.

4.3 Improvement 2: Hierarchical PDBs

Note that once there is a simple heuristic in hand, the new PDB heuristic only requires storing entries for patterns which their PDB values are larger than the simple heuristic. This suggests an hierarchy of PDBs. First, you store a small weak PDB. Then, for the stronger PDB you store only those entries having values larger than the weaker PDB. We used this idea as follows. For any 8-tile PDB we only need values which are larger than the 2-6 partitioning described above. Values of the 8-tile PDB which are not larger can be omitted and retrieved from the 6-2 PDBs. This was very effective as only 18% of the values of an 8-tile PDB were larger than the corresponding 6-2 PDB and had to be stored. The overhead for this was the need to generate and store the relevant 6-tile and 2-tile PDBs but we stored the 6-6-6-6 PDBs anyway as described above and the overhead of generating and storing a number of 2-tile PDBs is very low.

Here we only stored two levels using this hierarchical approach. Future work can take this further by building an hierarchy of PDB heuristics where each PDB is built on top of the previous one in the hierarchy. Note that the basics of this approach were used in [8] for the 15 puzzle where the weaker heuristic was MD and only additions above MD were stored in the PDB. Thus, values for patterns equal to MD were stored as 0. Here we further improve on this approach and omit values of 0.

4.4 Improvement 3: Multiple Partitioning

The bottleneck for the IDPDB method is the memory requirements of the secondary A* search. This phase terminates when memory is exhausted. The memory requirements for the primary search phase is much smaller especially after applying all the improvements above. It is well known that PDBs are better utilize by having multiple PDBs and taking the maximum value among them as the heuristic [6, 7]. Since so much memory was released we were able to use the extra memory for storing six different 8-8-8 partitionings illustrated in figure 5 and taking their maximum as the heuristic.

Table 1. ISPDB with 8-8-8 and 9-9-6 partitionings

Method	Nodes	Entries	Mem	Hits
1 6-6-6-6	4,756,891,097	255,024,000	244	100
2 6-6-6-6	1,107,142,063	255,024,000	244	100
1 Gigabyte				
OD	1,434,852,411	7,178,143	125	27.7
OD+6	938,256,516	7,178,143	369	28.9
PP	856,917,588	25,071,429	656	66.4
Imp1	714,722,200	25,071,429	656	68.5
Imp2	714,722,200	4,538,015	327	16.6
6 8-8-8	198,851,450	15,638,294	505	-
8 9-9-6	175,100,719	31,199,159	754	-
2 Gigabytes				
PP	713,536,979	66,690,152	1,322	80.6
Imp1	613,844,599	66,690,152	1,322	83.3
Imp2	613,844,599	13,565,100	472	21.2
6 8-8-8	130,890,131	48,907,720	1,037	-
8 9-9-6	100,964,443	82,143,861	1,569	-

5 Experimental Results

We implemented all the above variations and improvements on the same random instances of the 24 puzzle used by [8]. We experimented with the 8-8-8 partitionings of figure 5 and also with a set of 9-9-6 partitionings. We used a 1.7 MHz Pentium 4 PC with one gigabyte of main memory and also with two gigabytes. The primary search was performed with IDA*.

We sorted the 50 instances from [8] in increasing order of solution length. Table 1 presents the average results on the ten "easiest" instances. The first column indicates the variation used and the second column counts the number of generated nodes. The third column, *Entries*, is the total number of 8-8-8 (or 9-9-6) PDBs entries that were finally stored. The *Mem*, gives the total amount of memory in megabytes used for all the PDBs consulted by this variation (including the 6-6-6-6 PDB when applicable). Finally, the last column indicates the percentage of times where the 8-8-8 (9-9-6) PDB had a hit. We define a *hit* as a case where a PDB is consulted and actually had an entry for the specific pattern. This is opposed to a *miss* where that entry was not available and the simpler heuristics were consulted.

Often, a stronger heuristic consumes more time per node. Thus, the overall time improvement to solve the problem with a stronger heuristic is less than the reduction in the number of generated nodes. Nevertheless, the actual time is influenced by the effectiveness and effort devoted to the current implementation. For example, using a better hash function or sophisticated data structures for storing entries in the PDB might further improve the running time. A number of methods for reducing the constant time per node when using multiple PDBs lookups were provided by [6]. Using as many of these methods further reduces the overall time. The actual time also depends on the hardware and memory abilities of the machine used. We noted that the number of nodes

per second in all our variations was always between one to two Million. Since the nodes improvement reported below is significantly greater we decided to omit the time reports and concentrate only on the number of generated nodes. As discussed above, we can also omit the time of generating the PDBs which took between 30 to 80 minutes for our different variations. This is negligible when compared to the actual search time which was a around 18 hours on average for the random 50 instances.

The first row of table 1 uses one 6-6-6-6 partitioning. The second row is the benchmark results taken from [8] where the 6-6-6-6 partitioning was also reflected about the main diagonal and the maximum between the two was used. This reduces the search effort by a factor of 4.

In the next bunch of rows we had one gigabyte of main memory for the secondary A*. The first row (OD) is the simple case where only a single 8-8-8 partitioning (of figure 5.a) was used and the extended secondary A* search was performed on demand. In a case of a miss in a PDB, we calculated the Manhattan distance (MD) for the tiles in this particular PDB. Here, only 7,178,143 entries were generated since the open lists of the different 8-tile PDBs were stored in memory during the primary search. Note that the hit ratio here is low (27.7) as the size of the PDB is comparably small. Even this simple variation of one 8-8-8 partitioning reduced the number of generated nodes by more than a factor of three when compared to the one 6-6-6-6 partitioning.

The second row (OD+6) also generated the PDBs on demand. However, here, (and in all the successive rows) we took the maximum between the 8-8-8 and the 6-6-6-6 PDBs. This variation outperformed the one 6-6-6-6 version (line 1) by a factor of 5 and the benchmark two 6-6-6-6 version (line 2) by 18%. The next line (PP) is the preprocessing variation where the entire secondary search for each 8-tile PDB was performed a priori until memory was exhausted. Here more patterns were expanded by the secondary A* search and therefore we could load 25,071,429 values to the PDBs. This reduced the number of generated nodes to 856,917,588. Note that the hit ratio was increased to 66%. While the total number of patterns for three 8-tile sets is 129 Billion entries we stored only 25 Million (a fold factor of 4,600) and yet had the relevant value in 66% of the times.

The fourth line (Improvemnet 1) used the 6-2 partitioning instead of MD when a miss occurred. The fifth line (Improvement 2) only loaded the 8-tile values that are larger than the corresponding 6-2 partitioning. Here we can see a reduction of a factor of four in the number of stored entries. In both variations the number of generated nodes was reduced to 714,722,200. With improvement 2, the hit ratio dropped to 16.6 since many of the entries were removed as they were no larger than the 6-2 partitionings. Note that improvement 2 squeezed the PDB to a small size which enabled us to store multiple PDBs below.

In the next line full advantage of main memory was taken and six different 8-8-8 partitionings were stored. This reduced the number of nodes to 198,851,450. In the last line we used 8 9-9-6 PDBs. Here the number of nodes is 175,100,719, eight times smaller then benchmark results of two 6-6-6-6 partitionings. Here we did not report the hit ratio as it was difficult to define it for multiple lookups.

We then report similar experiments performed when we had two gigabytes of main memory for generating the PDBs. Here, more patterns were generated and the final

Table 2. Results over 25 and 50 instances

Method	Mem	Nodes	Ratio1	Ratio2
The 25 easiest instances				
2 6-6-6-6	-	16,413,254,279	1	1
8 9-9-6	1GB	3,788,144,197	4.33	7.37
6 8-8-8	1GB	2,897,728,901	5.66	6.59
8 9-9-6	2GB	2,339,671,729	7.01	12.85
6 8-8-8	2GB	2,037,614,978	8.05	10.25
All 50 instances				
2 6-6-6-6	-	360,892,479,671	1	1
6 8-8-8	2GB	65,135,068,005	5.54	8.85

PDBs were larger. Note that the hit ratio here increased to 83.3%. The number of generated nodes here was 130,890,131 for the multiple 8-8-8 PDBs and 100,964,443 for the 9-9-6 PDBs. This is approximately one order of magnitude better than the previous benchmark results for these 10 instances.

Our most successful method used six 8-8-8 and eight 9-9-6 partitionings. With this method we solved the "easiest" 25 instances (the first 25 from the sorted list of instances) with 1GB and 2GB of memory. Results are presented in Table 2. Since the number of generated nodes in the tile puzzle is exponentially distributed it is problematic to report average results. Therefore, we report two different numbers for the improvement factor over the 6-6-6-6 benchmark. The first number (*Ratio1*) is the ratio of the total number of nodes for the 25 instances. For the second number (*Ratio 2*) we calculated the improvement factor for each instance alone and report the average over all the 25 factors. When considering the total number of generated nodes over all instances, the 8-8-8 partitioning improved the benchmark results by a factor of 8 and when considering each instance alone the 9-9-6 reduced the number of nodes by a factor of almost 13.

Finally, we solved the entire set of 50 instances from [8] with 6 8-8-8 PDBs. It seems that the effectiveness of IDPDB drops a little for the more difficult instances. This is because there are more A* patterns but the PDB has the same size and therefore the chance for *missing* a pattern from the PDB increases. Still, even in difficult instances, IDPDB managed to focus on the relevant patterns and the total number of nodes over all 50 instances was 65,135,068,005 - 5.54 smaller than pervious benchmark results. On a single instance basis the improvement factor ranged from 3.5 to 40 and averaged 8.85. It took us about a month to solve all 50 instances. To the best of our knowledge we have the best published optimal solver for this problem.

6 Summary and Conclusions

We presented simplifications and enhancements to the instance dependent PDB method suggested by [10]. We showed that instead of the fixed mathematical set they computed, we can dynamically grow the PDB until memory is exhausted. With our method we

optimally solved random instances of the 24 puzzle by using 8-8-8 (9-9-6) disjoint partitionings. A single complete 8-tile PDB needs 43 Billion entries. We reduced the number of entries by a factor of 1000 and yet produced the state of the art performance for this problem.

For future work we intend to combine partitionings of different sizes e.g. an 8-8-8 with a 9-9-6 etc. We would also intend to build a larger hierarchy of different pattern databases each adds only relevant values to its predecessor. For example, given a 5-tile pattern database, we can use it for adding a sixth tile by only storing the contribution of this new tile to the 5-tile pattern databases. Also, a deeper mathematical analysis and a theory that unifies this work with the work of [10] should be introduced.

References

1. J. C. Cullberson and J. Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.
2. S. Edelkamp. Planning with pattern databases. *Proceedings of the 6th European Conference on Planning (ECP-01)*, 2001.
3. A. Felner, R. E. Korf, and Sarit Hanan. Addtive pattern database heuristics. *Journal of Artificial Intelligence Research (JAIR)*, 22:279–318, 2004.
4. A. Felner, R. Meshulam, R. Holte, and R. Korf. Compressing pattern databases. In *Proc. AAAI-04*, pages 638–643, 2004.
5. I. Hernádvölgyi and R. C. Holte. Experiments with automatically created memory-based heuristics. *Proc. SARA-2000, Lecture Notes in Artificial Intelligence*, 1864:281–290, 2000.
6. R Holte, J. Newton, A. Felner, and D. Furcy. Multiple pattern databases. *Proc. ICAPS-04*, pages 122–131, 2004.
7. R. E. Korf. Finding optimal solutions to Rubik's Cube using pattern databases. *Proc.AAAI-97*, pages 700–705, 1997.
8. R. E. Korf and A. Felner. Disjoint pattern database heuristics. *Artificial Intelligence*, 134:9–22, 2002.
9. M. McNaughton, P. Lu, J. Schaeffer, and D. Szafron. Memory efficient A* heuristics for multiple sequence alignment. In *Proc. AAAI-02*, pages 737–743, 2002.
10. R. Zhou and E. Hansen. Space-efficient memory-based heuristics. In *Proc. AAAI-04*, pages 677–682, 2004.
11. R. Zhou and E. Hansen. Structured duplicate detection in external-memory graph search. In *Proc. AAAI-04*, pages 683–689, 2004.

Combining Feature Selection and Feature Construction to Improve Concept Learning for High Dimensional Data

Blaise Hanczar

Lim&Bio, University Paris 13, Bobigny, France
hanczar@limbio-paris13.org

Abstract. This paper describes and experimentally analyses a new dimension reduction method for microarray data. Microarrays, which allow simultaneous measurement of the level of expression of thousands of genes in a given situation (tissue, cell or time), produce data which poses particular machine-learning problems. The disproportion between the number of attributes (tens of thousands) and the number of examples (hundreds) requires a reduction in dimension. While gene/class mutual information is often used to filter the genes we propose an approach which takes into account gene-pair/class information. A gene selection heuristic based on this principle is proposed as well as an automatic feature-construction procedure forcing the learning algorithms to make use of these gene pairs. We report significant improvements in accuracy on several public microarray databases.

1 Introduction

Transcriptomics is the description and analysis of data related to the study of gene profiles and expression. This area has made great progress in recent years particularly thanks to DNA chips (or microarrays). A growing number of bioscientific projects now include studies based on this technology because it allows simultaneous measurement of the expression of several tens of thousands of genes. Promising applications for these chips include their use for improving the diagnosis of certain diseases such as cancer and also for providing better understanding of their etiology [6]. In these applications, the role of classification is often crucial, and different approaches have been explored, including Bayesian Networks, Neural Trees, Radial Basis Function Neural Networks [11], Support Vectors Machines, k Nearest Neighbours and Diagonal Linear Discriminant.

The task of bioinformatics is therefore often to construct classifiers from genes expression where each patient is described by numerical values corresponding to the levels of expression of the genes represented on the microarray. The classifiers must predict as precisely as possible a clinical parameter (such as the type of tumour) representing the class. One of the problems in building classifiers from microarray data is the unbalance between the number of examples (patients) and the number of features (gene expression). Actually, the biggest dataset available

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 261–273, 2005.

in the literature include few patients (between 50 and a few hundred) and a large number of genes (from a few hundred to forty thousand). It has been demonstrated, in fact, that too large a number of dimensions favours overfitting; this is the problem known as "the curse of dimensionality" [1]. To overcome this problem, dimension-reduction methods are classically used in machine learning. The aim of this step is to identify a reduced subset of attributes which maximizes prediction performance. These methods are widely used in the microarray data analysis.

A characteristic of these data is known presence of possibly strong interactions among gene expression (features). Handling feature interactions is difficult because of their intrinsic combinatorial nature. For this reason, the majority of reduction methods take little or no account of interactions between genes. Detection of interactions is implicitly left to the learning algorithms downstream of the selection phase. In this paper we investigate the possible advantages of considering gene interactions in the phase of dimension reduction itself. Previous studies have shown that pairs of genes with high discrimination power are not usually constituted by genes which are both significantly very discriminant. Then, a feature reduction method that does not consider interactions explicitely is likely to miss the weakest element of "good" pairs. To overcome the mentioned problem, we have developed a feature-construction approach: each newly constructed feature synthesizes the information contained in pairs of strongly interacting genes.

The remainder of this paper is organized as follows. Section 2 presents the state of the art in dimension-reduction methods in the field of microarray data. Section 3 proposes a heuristic measure for gene pairs with strong information. Section 4 shows how to use this information in machine-learning algorithms. Section 5 contains the results of an extensive experimentation; we show the advantages of considering gene interaction for feature construction.

2 Related Work

There is a vast amount of work on gene selection methods to improve microarray data classification. They can be classified into three families: a) scoring methods, b) methods selecting subsets of genes and c) reformulation methods. The most common approach is the scoring methods, which consider each gene individually and link its expression with the classes. For each gene a relevance score is computed depending on how well the gene distinguishes the examples of different classes. A good review of this kind of methods was made by Ben-Dor [2]. Subset-selection methods do not consider genes individually but groups of genes. Whether or not a gene is selected depends on other genes. It is therefore not surprising to find that this family includes many techniques that come from machine learning, particularly genetic algorithms [15], wrapper methods [12], and SVM-RFE (Guyon). Reformulation reduction methods project the data into a new smaller space, which is defined by attributes which are a combination of genes. Principal-component analysis is the best known of the methods in this family.

Other methods of changing representations specifically developed for microarray data also exist, including Qi's method [16], based on the amplitude and statistical form of genes for constructing new features. The ProGene algorithm [10] is another method belonging to this family; it creates gene prototypes to compress the information contained in groups of genes with similar expressions.

Most of these dimension-reduction methods do not take explicitly into account the interaction between genes. Nevertheless, some research has proposed approaches using gene interactions, based on the assumption that genes with similar expression provide redundant information. For example Xing [18] and Wu [17] compute a relevance score for each gene, then select a subset which maximizes the sum of relevance scores and minimizes the redundancy between the selected genes. All these methods explore individually the information contained in each gene and then try to find a good combination. Previous methods considering gene pairs also exist. Bo [3] evaluates a pair by computing the projected coordinates of each example on the DLD axis in the gene-pair space. The score is the two sample t-statistic on the projected points. Geman [9] does not use the expression value, but the expression rank of genes. The pair score is based on the probability of observing that the rank of the first gene of the pair is higher than the second one in each class. Their experimental results confirm the claim that class prediction can be improved using pairs of genes.We propose in this paper to identify highly interacting gene pairs, and systematically exploit these synergies to improve classification accuracy.

3 Reducing Dimensions by Using Higher Order Gene Information

3.1 Definition of *Gene Information*

To quantify the information that a gene subset provides in order to predict the class, we use a general measure of information I [13], which is based on entropy H and defined by the following formula:

$$I(X) = H(C) - H(C|X)$$

Where C is the class to be predicted and $X = \{G_1, ..., G_p\}$ a subset of genes. This information measure is the decrease in the entropy of the class brought by the gene subset. When the subset does not contain any information, the measure is minimum i.e. $(I(C) = 0)$. When the subset eliminates all uncertainty about the class, the measure is maximal i.e. $(I(C) = H(C)))$. When the information of only one gene is evaluated, this measure is equivalent to mutual information.

$$I(G) = H(G) + H(C) - H(G, C)$$

$$H(G, C) = H(C|G) + H(G)$$

The information of a gene subset can be broken down into a sum of information of each gene plus the *interaction information* contained in gene interaction

[13]. Following Jakulin's definition, the gene information for a pair of genes is defined as follows :

$$I(G_1, G_2) = I(G_1) + I(G_2) + Interaction(G_1, G_2)$$

$$\begin{aligned} Interaction(G_1, G_2) = &-H(G_1, G_2, C) + H(G_1, C) + H(G_2, C) \\ &+H(G_1, G_2) - H(G_1) - H(G_2) - H(C) \end{aligned}$$

Unlike the gene information, *interaction information* may be negative. When this interaction is positive, we talk of *synergy* between the genes, and *redundancy* otherwise.

3.2 The Search for the Most Informative Pairs of Genes

Our objective is to find the pairs of genes which provide the highest information. To identify the optimal pairs of genes, all of the gene-pair space of size N^2 must be explored, where N is the number of genes. In the context of microarray data where the number of genes is of the order of several thousands, this solution is often computationally difficult to adopt. To avoid exploring all of this space, a natural heuristic for exploring the pairs consists of first calculating the information of each gene and then calculating the information of the N-1 gene pairs formed by the best gene and another gene. The most informative gene pair is selected and the both genes of this pair are removed from the list of genes. The process of selection is then iterated to increase the number of selected genes. To find p pairs of genes, $(2N(p-1) + p^2 + 1)$ pairs only need to be explored. Another advantage of this heuristic is to generate pairs formed from *distinct* genes. It is well known in machine-learning literature that redundancy in features has a negative influence on classification [5]. Since we assume that pairs containing the same gene are likely to be redundant, we have chosen the above mentioned heuristic to search for the more informative pairs.

3.3 Feature Construction from Gene Pairs

The use of synergic pairs of genes does not necessarily improve prediction accuracy over a selection method that selects individual genes based on their mutual information with the class. The results given later in table 3 are an example of such a case. The reason is that the used learning algorithms were not designed to exploit pairs of features, so that they don't necessary use the synergy contained in gene pairs. We therefore propose an approach based on feature construction for synthesizing the information contained in each synergic pair. The FEATKNN method that we have developed to construct new features is limited here to two-class problems. Its adaptation to multi-class problems is beyond the scope of this paper.

For each pair P of synergic genes a new feature A taking its values between -1 and 1 is constructed. This new feature should capture as much information

Algorithm 1. Reduction dimension and feature construction algorithm

1. $Features \leftarrow \oslash$
2. $Genes \leftarrow all.gene$
3. for i from 1 to nb.pair.max
   (a) Search of gene pair G1 G2
      i. $G1 \leftarrow argmax_g\,(I(g))$
      ii. $G2 \leftarrow argmax_g\,(I(G1, g))$
   (b) Construction of a new attribut F
      i. for j from 1 to nb.sample
         A. $N \leftarrow$*k nearest neighbours of sample j*
         B. $n_+ \leftarrow$*number of sample of class+ contained in N*
         C. $F[j] \leftarrow -1 + 2\frac{n_+}{k}$
   (c) $Features \leftarrow Features \cup F$
   (d) $Genes \leftarrow Genes\text{-}\{G1, G2\}$
4. return (Features)

as possible of the two genes and their interaction. Building a new feature A requires defining its values based on the values of the two genes G_1G_2 of the pair P. A may then be described as a function from the two dimensional space $G_1 \times G_2$ defined by G_1 and G_2 to the intervale [-1,1]. Our rationale for defining such function is that it should capture in this space the density of the respective positive and negative examples.

The new features are constructed using the following method. Given an example t, the value of its feature A is computed as follows: $A(t) = -1+2\frac{n_1}{k}$ where n_1 is the number of k nearest neighbours of t belonging to class 1. The distances are computed in the $G_1 \times G_2$ space using the Euclidian distance. Note that t is not counted among the k nearest neighbours, and then its class does not intervene in the feature construction. The new feature A has values in [-1,1]. When A equals 1 (resp. -1), it means that almost all k neighbours of t in the $G_1 \times G_2$ space are labelled class 1 (resp. class 2). The number k of nearest neighbours is an important parameter, it controls the smoothing of the new attributes. If k is too small the risk of overfitting is high, and if K is too large the new attributes will be completely smooth and will have about the same value over the whole pair space. This problem is similar to the classical dilemma between variance and bias in classification problems. After the experiments, we defined $k = \frac{n}{5}$ as a good trade-off, where n is the number of examples. Figure 1 shows an example of feature construction with the best gene pair from the colon cancer dataset, varying the value of k. In the down panel (k=40) the new feature depends only on the gene Hsa.37937, the second gene information is not used. In the up panel (k=2) we see a situation of overfitting. The centre panel (k=12) presents a good trade-off. The whole dimension-reduction procedure that integrates both gene selection and feature construction is described in Algorithm 1.

Another method of feature constructed was developed, called FEATHR. The example are projected in the two dimensional space defined by the two genes of

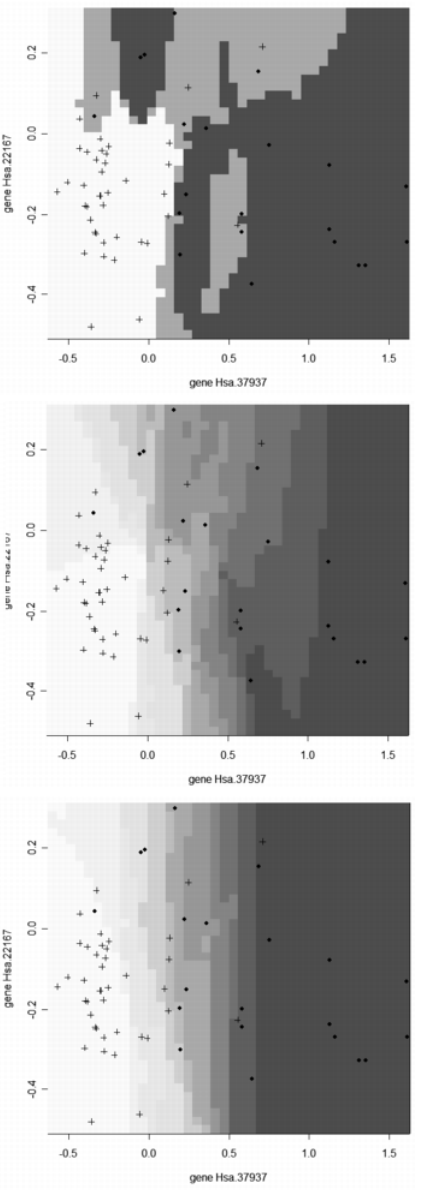


Fig. 1. Illustration of three alternative features using different values of k (cf. Algorithm 1 line C), k=2 (up) k=12 (centre) k=40 (down) and constructed with the best gene pair from the colon cancer dataset (Hsa.37937 and Hsa.22167). Sick patients (crosses) and safe patients (black dots) are represented in this gene-pair space. The grey level represents the value of the newly constructed feature (white representing a null value and black a value of one). The feature built with k=12 offers the best learning results. A good value of the hyper-parameter k may be empirically searched

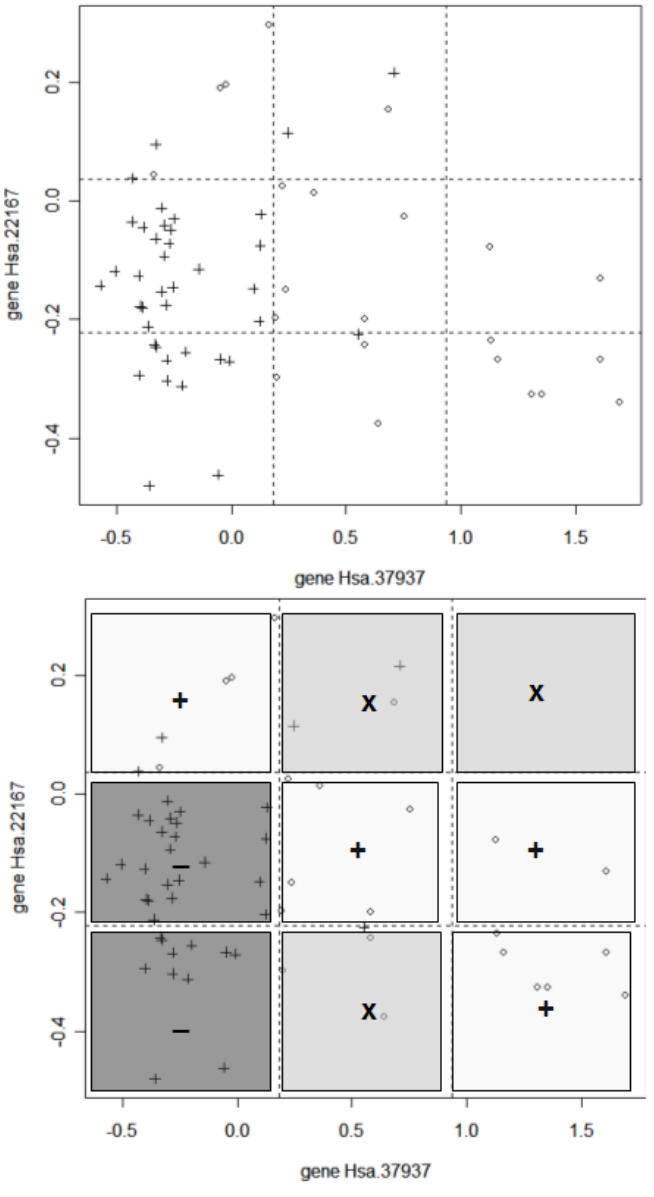


Fig. 2. Illustration of features constructed by FEATHR with the best gene pair from the colon cancer dataset (Hsa.37937 and Hsa.22167). Sick patients (crosses) and safe patients (black dots) are represented in this gene-pair space. The up panel presents the patients projected in the two dimensional space of the gene pair. In this space 9 sectors were created. The down panel presents the value of the new feature associated with each sector. The new feature, associated with sectors contained only examples of class "sick" (resp. "safe"), takes the values "-" (resp. "+"). In a sector where examples of the two classes are present the new feature takes the value "x"

each pair. This space is divided in sectors of equal size. The value of the new feature is the same for all points in the same sector. The value of the new feature depends on the presence of the examples belonging to class 1 (or -1) in the sector. In this case, the new feature is therefore a discrete feature. Figure 2 illustrates this method. We obtain relatively bad performances with this method, that is the reason we did not present its results in the experimentation section 5.

4 Experimentation

The experimental study is designed to anwers the following questions:

1) *Is our selection heuristic adapted to find informative pairs of genes?*
2) *Is our feature construction method effective to synthesize the information contained in a pair of genes?*
3) *Does our dimension-reduction method improve the classification accuracy as opposed to classical methods using mutual information and other methods exploring gene pairs?*

4.1 Data

Two different datasets are used in these experiments, where characteristics are illustrated in table 1 . To compute information measures, expression data were discretized. With to the biologists, we assume that the gene expression can be in one of three states: overexpressed, non-modulated, underexpressed. We use a histogram method to discretize the expression of each gene in the three states. The amplitude of the gene expression is computed, then divided into three subintervals of equal size.

Table 1. Description of the datasets. This table shows the data type, the number of genes measured and the number of samples contained in each class

	#genes	#samples by classes
Leukemia	7129	47 ALL / 25 AML
Colon Cancer	2000	40 sick patients /22 safe patients

4.2 Analysis of the Most Informative Pairs

In order to measure the importance of the interactions between genes, we empirically examined the mutual information of the best genes and pairs of the colon cancer data set. The data contained relatively few genes (2000); it was therefore possible to calculate the information of all the gene pairs, i.e. 1,998,000 pairs. A rank based on information measure is thus defined for each gene and each pair of genes. Table 2 shows the rank and information measure of the 20 best gene pairs of the colon cancer dataset. For example, we see that the best gene pair is formed by the best gene (Hsa.37937) and the 592nd best gene (Hsa.22167).

Table 2. The 20 best gene pairs for colon cancer. For each gene pair (g1,g2), the information (I) and the rank (R) of each of the two genes of the pair, the interaction (inter), the information of the pair and the newly constructed feature I(F) were computed. The pair are ordered by their information I(g1,g2))

	g1	I(g1)	r(g1)	g2	I(g2)	R(g2)	inter	I(g1,g2)	I(F)
pair 1	Hsa.37937	0.47	1	Hsa.22167	0.06	592	0.26	0.79	0.35
pair 2	Hsa.8147	0.38	2	Hsa.3933	0.08	355	0.16	0.62	0.34
pair 3	Hsa.934	0.13	146	Hsa.1131	0.3	4	0.19	0.62	0.2
pair 4	Hsa.25322	0.28	5	Hsa.36696	0.2	33	0.13	0.61	0.36
pair 5	Hsa.22762	0.2	40	Hsa.7	0.26	9	0.14	0.6	0.47
pair 6	Hsa.579	0.22	23	Hsa.5392	0.13	135	0.22	0.57	0.25
pair 7	Hsa.878	0.25	11	Hsa.442	0.15	95	0.17	0.57	0.32
pair 8	Hsa.6376	0.05	750	Hsa.1832	0.34	3	0.02	0.41	0.26
pair 9	Hsa.6814	0.17	63	Hsa.2939	0.17	61	0.22	0.56	0.2
pair 10	Hsa.1517	0.01	1583	Hsa.127	0.14	109	0.4	0.55	0.22
pair 11	Hsa.812	0.14	123	Hsa.2451	0.24	13	0.17	0.55	0.35
pair 12	Hsa.3305	0.24	15	Hsa.466	0.2	34	0.1	0.54	0.26
pair 13	Hsa.42949	0.09	315	Hsa.2928	0.18	51	0.27	0.54	0.23
pair 14	Hsa.821	0.23	22	Hsa.43431	0.06	542	0.25	0.54	0.27
pair 15	Hsa.8068	0.18	59	Hsa.1317	0.18	54	0.18	0.54	0.14
pair 16	Hsa.2386	0.07	474	Hsa.692	0.27	8	0.2	0.54	0.35
pair 17	Hsa.36694	0.19	49	Hsa.1276	0.11	218	0.23	0.53	0.19
pair 18	Hsa.1682	0.13	136	Hsa.21868	0.07	434	0.33	0.53	0.19
pair 19	Hsa.692	0.27	8	Hsa.31801	0.13	138	0.12	0.52	0.21
pair 20	Hsa.41280	0.23	18	Hsa.18787	0.02	1248	0.2	0.45	0.32

Figure 3 shows the information of genes forming the best first million pairs; each point represents a set of 10,000 pairs. The black dot corresponds to the gene with the highest information of the pair and the white dot corresponds to the other gene of the pair. We see that the best pairs are on average all formed by a highly informative gene and a low informative gene. This property of highly informative gene pairs needs to be proved experimentally. This observation allows us to give a positive answers (from an empirical point of view) to the first question regarding the ability of our heuristic to find highly informative gene pairs. It also demonstrates the usefulness of our heuristic choice for selecting pairs, which are formed from genes with the best rank.

4.3 Information Obtained by Feature Construction

The aim of feature construction is to synthesize the information contained in the genes and their interactions. In order to measure the importance of feature construction, we empirically compare the information of the genes with the newly constructed feature on the colon cancer dataset. Figure 4 shows this comparison; dots represent the information of genes and crosses the information of gene pairs. The solid line represents the case where the information of newly constructed feature of gene pair would be strictly equal to the gene (or gene pair)

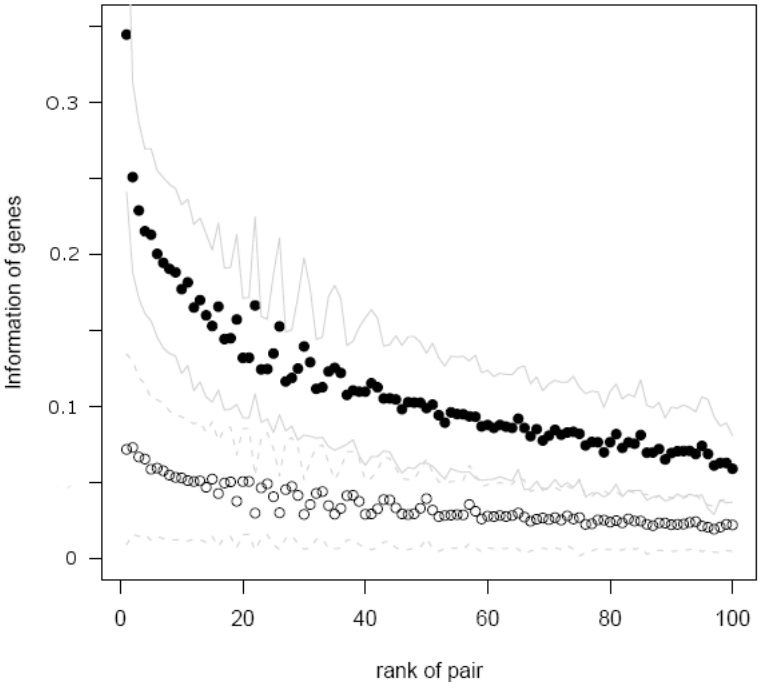


Fig. 3. Information of the two genes of the first million best pairs; each dot represents the average value of a set of 10000 pairs. A black dot corresponds to the average of the gene with the highest information of the pair whereas the white dot corresponds to the average of the second one. The two thin symetric lines above each set of dots represent the standard deviation of the information value of the 10000 genes composing each set

information. Most circles are on the left of the solid line, which means the newly constructed features are more informative than the genes they are derived from. Those results show that our feature construction method is effective for synthesizing the information contained in a gene pair, answering our second question. In constructing these new features the information contained in two genes is compressed into only one feature, It is not surprising that this new feature does not automatically capture all the information of the pair.

4.4 Classification Accuracy

We have compared our method to the classical method using mutual information and to the gene-pair based methods of Geman and Bo. The classical method using mutual information is a scoring method where the mutual information between each gene and the class is computed individually. Genes which have the higher correlation with the class are selected. The complete description of the methods of Geman and Bo can be found in the bibliography [9, 3]. In order to measure the impact of these methods on classification, we examined classi-

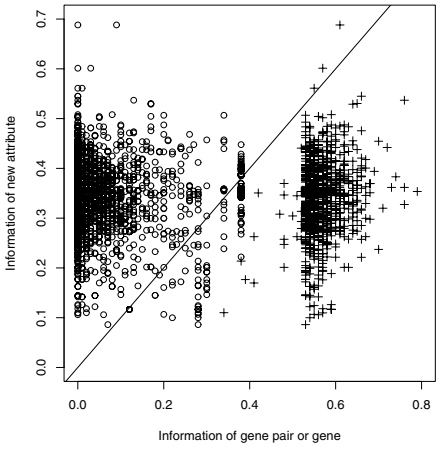


Fig. 4. Comparison of the information of newly constructed features and information of gene (dot) or gene pair (cross). The solid line represents the case where the newly constructed feature information of a gene pair would be strictly equal to the gene (or gene pair) information

Table 3. Different classification results on the two public datasets. All errors are estimated using the .632 bootstrap estimators

Algo	Data	All Gene	Info. Mut.	Gene Pairs	FeatKnn	Bo	Geman
SVM	Leukemia	12.3	4.3	4.8	**2.8**	3.9	6.1
	colon c.	17.5	12.5	11.8	**10.7**	13.9	14.6
KNN	Leukemia	8.4	4.8	6.2	5	**4.6**	6.3
	colon c.	17.5	13.9	14.4	**12.8**	15.9	16
DLD	Leukemia	11.5	4.8	4.8	**3.8**	4.1	5
	colon c.	19.5	14.7	15.4	**12.5**	14.4	15

fication accuracy on two datasets. First the informative gene pairs were identified, secondly new features were constructed, then a classifier was built by the classification algorithm and finally the generalization error of the classifier was computed. It should be noted that the dimension-reduction step was performed within the evaluation procedure and not before. The cross-validation estimator (particularly the 10-cross validation or Leave-On-Out) is commonly used to compute the generalization error. On the two public databases we have used, to the best of our knowledge the best results obtained are 0% error rate on the leukemia dataset and 10.7% on colon cancer dataset using the Leave-On-Out estimator. However Braga-Neto showed that this estimator is not the most appropriate one for a small sample context like microarrays [4]. Cross-validation has a high variance and bootstrap estimators are preferred, in particular the .632

estimator [8]. This is a weighted sum of the empirical error and the out-of-bag bootstrap error (100-bootstrap iterations were performed) and is the method we have chosen to evaluate the classification accuracy. We used three classification algorithms: support vector machines (SVM), k-nearest neighbours (KNN) and diagonal linear discriminant (DLD). The three are accurate in classifying microarray data [7, 14].

Table 3 summarizes the different classification performance of the algorithms and reduction methods. It is not surprising to see that dimension-reduction methods considerably improve classification performance, and the methods of selecting the best genes and best pairs give similar results. How can we explain why these pairs do not improve classification performance? It is probable that the information contained in the interaction between the genes of one pair is not totally exploited by the classification algorithms, and much of the information computed during the pair-selection phase is thus lost. The new features constructed by FEATKNN synthesize the information contained in the genes and their interactions and in this case the classification exploits the interaction between the genes using these newly constructed features, which explains the better results. FEATKNN outperforms the other two methods of Bo and Geman, both of which give results that are similar to those of gene pairs. We can suppose that Bo's and Geman's methods find relevant gene pairs, but the classification method can't exploit their information completely. The fact remains nonetheless that biological interpretation becomes different from that of the classic approach. There are no longer any maximally discriminating genes but lists of pairs, which it might be possible to use in the study of regulation networks.

5 Conclusion

In this paper we have presented a dimension-reduction procedure for microarray data oriented toward improving classification performance. These methods are based on the hypothesis that the information provided by the interaction between genes cannot be ignored in the feature selection phase. We have limited this study to interactions between pairs of genes. Although it is natural to quantify information from genes and interactions from the computation of mutual information, this simple reduction does not necessarily improve performance. Thus, we have developed a feature construction method, FEATKNN, which forces learning algorithms to take into account pairs with a high level of mutual information. The experimental usefulness of these interactions was assessed on two datasets where performance was improved. We are currently systematically analyzing other microarray datasets to accumulate evidence of such an improvement. To analyze the biological interest of gene pairs we are also working on as we have biological experts that may analyze the pairs.

Another research direction is aimed at taking into account synergies between wider groups of genes, as well as theoretical analysis of the gains obtained by these dimension-reduction approaches.

References

1. R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.
2. A. Ben-Dor, N. Friedman, and Z. Yakhini. Scoring genes for relevance. Technical Report AGL-2000-13, Agilent Technologies, 2000.
3. T. Bo and I. Jonassen. New feature subset selection procedures for classification of expression profiles. *Genome Biology*, 2002.
4. U.M. Braga-Neto and E. Dougherty. Is cross-validation valid for small-sample microarray classification? *Bioinformatics*, 20(3):374–380, 2004.
5. D. Cakmakov and Y. Bennani. *Feature selection for pattern recognition*. 2002.
6. Clement. Monogenic forms of obesity: From mice to human. *Ann Endocrinol*, 2000.
7. S. Dudoit, J. Fridlyand, and P. Speed. Comparison of discrimination methods for classification of tumors using gene expression data. *Journal of American Statististial Association*, 97:77–87, 2002.
8. B. Efron. Estimating the error rate of a prediction rule: Improvement on cross-validation. *Journal of American Statistical Association*, 78:316–331, 1983.
9. D. Geman, C. D'Avignon, D. Naiman, R. Winslow, and A. Zeboulon. Gene expression comparisons for class prediction in cancer studies. *Proceedings 36'th Symposium on the Interface: Computing Science and Statistics*, 2004.
10. B. Hanczar, M. Courtine, A. Benis, C. Henegar, K. Clément, and J.D. Zucker. Improving classification of microarray data using prototype-based feature selection. *SIGKDD Explorations*, 5:23–30, 2003.
11. K.B. Hwang, D.Y. Cho, S.W. Park, S.D. Kim, and B.T. Zhang. Applying machine learning techniques to analysis of gene expression data: Cancer diagnosis. In *Methods of Microarray Data Analysis (Proceedings of CAMDA'00)*, pages 167–182. Kluwer Academic Publichers, 2002.
12. I. Inza, B. Sierra, R. Blanco, and P. Larrañaga. Gene selection by sequential wrapper approaches in microarray cancer class prediction. *Journal of Intelligent and Fuzzy Systems*, pages 25–34, 2002.
13. A. Jakulin and I. Bratko. Analyzing attribute dependencies. *Proceedings A of the 7th European Conference on Principles and Practice of Knolegde Discovery in Databases (PKDD)*, pages 229–240, 2003.
14. Jae Won Lee, Jung Bok Lee, Mira Park, and Seuck Heun Song. An extensive comparison of recent classification tools applied to microarray data. *Computational Statistics and Data Analysis*, In press.
15. L. Li, T.A. Darden, C.R. Weinberg, A.J. Levine, and L.G. Pedersen. Gene assessment and sample classification for gene expression data using a genetic algorithm/k-nearest neighbor method. *Combinatorial Chemistry and High Throughput Screening*, pages 727–739, 2001.
16. H. Qi. Feature selection and knn fusion in molecular classification of multiple tumor types. *International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences (METMBS'02)*, 2002.
17. X. Wu, Y. Ye, and L. Zhang. Graphical modeling based gene interaction analysis for microarray data. *SIGKDD Exploration*, 5:91–100, 2003.
18. E.P. Xing, M.I. Jordan, and R.M. Karp. Feature selection for high-dimensional genomic microarray data. In *Proceedings of the Eighteenth International Conference in Machine Learning, ICML2001*, 2001.

A Qualitative Spatio-temporal Abstraction of a Disaster Space

Zina M. Ibrahim and Ahmed Y. Tawfik

School of Computer Science,
University of Windsor, Ontario, Canada N9B 3P4
{ibrahim, atawfik}@uwindsor.ca

Abstract. This paper presents an abstraction of a vague and rapidly-changing environment, an urban disaster space, and a reasoning engine which recognizes and describes the motion of rescue agents as they traverse the disaster space. More specifically, we present a qualitative abstraction of the Robocup Rescue[1] simulation environment, and implement a *commentator* engine, which constructs a qualitative representation of the changes in the environment and produces descriptions of the agents' motion by recognizing patterns which motion can take. The patterns recognized are elements of a set of formalizations that qualify each type of motion pattern based on a qualitative representation of vague spatiotemporal moving objects. We also present a set of performance metrics to evaluate our qualitative representation.

1 Introduction

Qualitative spatial reasoning (QSR)[3] focuses on providing formal qualitative abstractions and reasoning frameworks that capture essential spatial properties as well as the relationships that hold among spatial regions. To reason about motion and spatial changes over time, qualitative spatiotemporal representations (QSTR) have been developed. Knowledge representations developed during the last two decades [14] have established the foundations for QSTR. Currently, the applications of QSTR include scene interpretation in cognitive vision [4], natural language understanding [15] , design tasks for engineering disciplines [8], GIS [6, 13] and robotics [7, 10].

In this paper, we examine the application of QSTR to the abstraction of an urban environment following a natural disaster. Frequently, rescue efforts are hampered by the lack of coordination among rescue agents. Qualitative abstractions of the dynamic changes in the disaster space provide the base for effective automated support for rescue efforts. One of our goals is to assess the advantages of representing disaster-world objects using a *qualitative spatiotemporal theory of vague objects*. As information from the rescue agents becomes available, a *commentator* agent generates qualitative description of the their motion.

[1] http://robot.cmpe.boun.edu.tr/rescue2004/

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 274–281, 2005.

The commentator qualitatively describes developments in the RoboCup Rescue simulation environment where there are fires, road blockades, civilian injuries and building destruction following a natural disaster. A number of independent software agents are responsible for rescue efforts in the city after the disaster.

The RoboCup Rescue simulation represents a dynamic environment where it is important to capture spatiotemporal changes. As the situation changes continuously, available knowledge is always *vague*. Using a spatiotemporal representation for vague regions, we develop qualitative abstractions of the environment. Objects including buildings and roads are used to create a new type of objects, *disaster-space* objects, which are abstractions of the real world objects. For example, a group of adjacent burning buildings are abstracted to create a fire burnout cluster, which is a vague region. Hence, the *disaster-space* objects can be represented by our qualitative theory for spatiotemporal regions with vague boundaries [9].

We compare the qualitative representation of the rescue environment to the existing quantitative one using three different parameters that we define.

2 Overview of the RoboCupRescue Simulation

The RoboCup Rescue project simulates a large urban disaster which takes place in a city where an earthquake has hit causing great damage. The city is two-dimensional and contains mobile as well as immobile objects whose locations are identified by their rectangular coordinates. The immobile objects include buildings and roads, while the mobile objects include the civilians and rescue agents. The disaster is simulated by three sub-simulators, each responsible for generating one type of after-effects of the earthquake. They include a fire simulator, a collapse simulator, and a blockade simulator.

When the disaster hits, the effects are reflected on the city and the agents in it. Some of the buildings sustain damage or fall blocking the roads. Fires break out in many buildings causing them to burn. Roads may become blocked due to the fallen debris. Agents may get injured or buried under the debris. The sub-simulators are responsible for deciding the extent of the damage to the world objects depending on their locations. The simulation contains a number of rescue agents whose main task is to clear the effects of the disaster and minimize the damage in the city. The rescue agents are of six types representing fire fighters, police officers, ambulance teams, fire stations, police centers and ambulance centers.

Hence, one can look at the RoboCupRescue environment as being made of three components, the disaster space, the sub-simulators and the rescue agents. In our work we have created a qualitative representation of the disaster space that we used to describe the motion of the rescue agents around objects of this representation. The agents creation, initialization and such tasks are part of the Robocup Simulation project and do not constitute any contribution for this work, hence they will not be discussed. Next, we review the main features of our qualitative spatiotemporal theory used in this work.

3 A Qualitative Theory of Vague Spatio-temporal Objects

In [9] we presented a formalization of the notion of motion of vague spatio-temporal objects. The aim of the theory is to identify the patterns that motion between two vague spatiotemporal objects can take, in order to formalize the notion of motion of regions with ill-defined boundaries and qualify its forms.

In [9], a moving object whose boundaries cannot be easily determined is the building block of the representation. We represent such object by an EggYolk pair [2] which is made of two concentric subregions, a white and a yolk as shown in figure 1. The yolk represents the parts that definitely belong to the region while the white represents the parts that may or may not belong to the region.

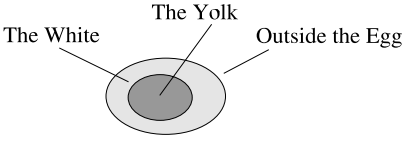


Fig. 1. The EggYolk Structure

Our vague moving object possesses topological properties that describe its spatial relations with other objects during a given period and temporal properties that describe the relations among the regions' lifetimes. The two types of properties jointly define and give the regions spatio-temporal properties.

We define the set of motion classes MC_v which represents the set of patterns of motion that may hold between two EggYolk spatiotemporal regions.

$MC_v = \{REACH_v\ z\ x\ y,\ LEAVE_v\ z\ x\ y,\ INTERNAL_v\ z\ x\ y,\ EXTERNAL_v z\ x\ y,\ P\_REACH_v\ z\ x\ y,\ P\_LEAVE_v\ z\ x\ y,\ P\_INTERNAL_v\ z\ x\ y,\ P\_EXTERNAL_v\ z\ x\ y\}$

Each $mc_v i\ z\ x\ y \in MC_v$, where $1 \leq i \leq 8$, is read as: motion pattern $mc_v i$ holds between vague (EggYolk) spatiotemporal regions x and y during interval z. For example, $REACH_v\ z\ x\ y$ is read as: vague spatiotemporal region x has reached vague spatiotemporal region y during interval z.

The elements of MC_v are divided into two subsets. The first includes patterns having no prefix, which represent complete occurrence of the motion class. The second includes patterns starting with prefix $P\_$ denoting partial occurrence of the motion class. In figure 2, the horizontal axis represents the spatial extent of the region and the vertical axis represents the temporal evolution of the region. Hence, the beginning of interval z is located in its lowest t value and its end is located at its highest t value. The spatial extents of regions x and y evolve as time increases and this forms the patterns shown in the figure.

Hence, in this work, we describe the vague rapidly-changing disaster-space objects (e.g. fire burnouts) as EggYolk spatiotemporal objects and use elements MC_v to formalize the motion of rescue agents around them.

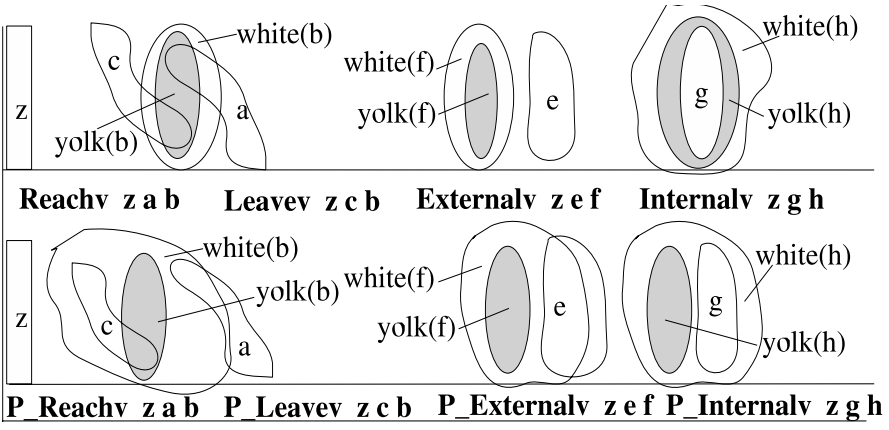


Fig. 2. MC_v Motion Classes

4 A Qualitative Commentator

Here we outline the structure of a commentator which produces the motion patterns formed as the rescue agents move from one EggYolk region to another.

4.1 Ontological Primitives

We create EggYolk spatio-temporal regions out of every situation where vagueness may exist. This is equivalent to creating a new form of objects which span through the real-world objects and abstract them to form vague spatio-temporal regions.

The difference between the real-world objects and their EggYolk equivalents is that the real-world objects have well-defined boundaries, size and location while the disaster-space objects are regions with vague boundaries whose size and location undergo continuous change as long as the simulation is running.

We have identified two types of EggYolk regions. The first is an abstraction of the buildings on fire and the second is an abstraction of blocked roads.

A **fire region** is made of a group adjacent buildings that are on fire. A *FireCluster* object *fc* is an EggYolk object whose yolk contains a fire region *fr* and whose white contains the buildings surrounding the fire region *fr*.

A **blockade region** is made of a group adjacent roads that are blocked. A *BlockadeCluster* object *bc* is an EggYolk object whose yolk contains a blockade region *br* and whose white contains the roads surrounding the blockade region *br*.

Hence, fire is no longer seen in terms of burning buildings, but in terms of EggYolk objects, where each is treated as a separate fire cluster. Although a cluster is made of buildings, these buildings are completely invisible to our representation and commentator logic and are treated as implementation details. The same picture can be drawn for the blockades as the only visible objects are the road clusters while the physical roads are invisible.

4.2 Motion Patterns

We define a set of capabilities that implement the elements of the set MC_v. More specifically, for each $mc_v i \in MC_v$, we implement the function:

$r = mc_v i$*(EggYolk ey, Agent ag, int* t_1*, int* t_2*)*

r is a boolean value that holds true if the motion pattern $mc_v i$ between *ey* and *ag* holds during the interval $[t_1,t_2]$.

4.3 The Commentator

The commentator maintains the lists *FAgents*, *PAgents*, *FClusters*, *BClusters* representing the sets of fire brigade agents, police force agents, EggYolk fire clusters and EggYolk blockade clusters respectively. The commentator object is created when the simulator starts at the same time the agents are created and simultaneously populates *FAgents* and *PAgents* from the simulation kernel. *FClusters* and *BClusters* are also created but are initially empty because the damage has not struck yet.

After the beginning of the simulation, the commentator performs a repetitive task every x cycles[2] which includes updating the lists *FClusters* and *BClusters* to reflect its new view of the world, and describing the motion patterns as follow:

r_f=[r_{f_k}|$\forall$ *fa*$\in$*FAgents*, $mc_v i \in MC_v$, $fc_l \in$*FClusters*, $r_{f_k} = mc_v i$(*fc_l,fa,t-x,t*)].
r_b=[r_{b_n}|$\forall$ *pa*$\in$*PAgents*, $mc_v j \in MC_v$, $bc_l \in$*BClusters*, $r_{b_n} = mc_v j$(*bc_l,pa,t-x,t*)].

Using lists r_f and r_b, the commentator produces descriptions of the motion patterns of the agents in *FAgents* as they traverse the *FireCluster* objects in list *FClusters* and agents in *PAgents* they traverse the *BlockadeCluster* objects in list *BClusters*.

5 Sample Run of the Commentator

Output is produced every 5 cycles. We choose as an example the output at an arbitrary cycle (cycle 25).

Since the aim is only to show the output language of the commentator, we chose to truncate parts of the output in order to preserve space.

```
Time = 25 World Description:
  World Objects: 45 Buildings and 48 Roads
  Agents: 7 Fire Brigades and 6 Police Force Agents
  15 Buildings on Fire, 3 Fires, 40 Blocked Roads, 10 Blockades
  Quantitative Description of Motion:
        Fire Brigade Agent 25 in in building b:8 - putting out fire
        Fire Brigade Agent 12 in in building b:19 - moving
```

[2] The cycle is the time unit used by the simulation, which runs for 300 cycles every time.

```
      Police Force Agent 91 in road r:13 - moving
      Police Force Agent 23 in road r:61 - clearing
Qualitative Description of Motion:
      Fire Brigade Agent(s) 25 reach cluster fc:1 - putting out fire
      Fire Brigade Agent(s) 12 possible_leave cluster fc:1 - moving
      Police Force Agent(s) 91, 17 internal cluster bc:3 -
                  17 clearing, 91 moving
      Police Force Agent(s) 63 possible_internal to cluster bc:9 -
                  clearing
```

6 Experiments and Results

We propose three performance metrics to evaluate our qualitative representation against the existing quantitative one. We use the commentator to perform our experiments and collect the average results of running the simulation 20 times on 3 different maps for both the qualitative and quantitative simulations.

6.1 Compactness

Measures the extent to which the representation minimizes redundancy in the knowledge it captures. A compact representation contains only the ***objects*** relevant to the tasks it ought to perform and ***events*** that cause a change to its knowledge base.

We define two variables, ***world_size*** which is the number of relevant regions the representation captures and the ***event_rate*** which is the number of events that take place during interval [*t-5*, *t*].

size_qual = *count(FClusters)+count(BClusters)*
size_quan = *count(world.burningBuidings)+ count(world.blockedRoads)*
event_rate_qual = $n_{event_{fire}}+n_{event_{block}}$
event_rate_quan = $p_{event_{fire}}+p_{event_{block}}$

Where $n_{event_{fire}}$ is the number of fire agents $fa_i \in FAgents$ traversing *Fire_Cluster* region $fc_j \in FClusters$.

$p_{event_{fire}}$ is the number of fire agents $a_i \in FAgents$ that change location from b_j to b_k where b_j, $b_k \in$ world.burning Buildings[3].

As the results in figure 3 show, the number of relevant events and objects generated in the qualitative experiments is less than the quantitative one.

6.2 Merit

Measures the precision at which the representation captures the world and its ability to discard false statements from its knowledge.

We identified two situations whose existence is considered a false representation of the world and accordingly formed two metrics. The ***percent exclusion***

[3] world.burningBuildings can be directly implemented in the existing simulator and used to extract the buildings of the world that are on fire.

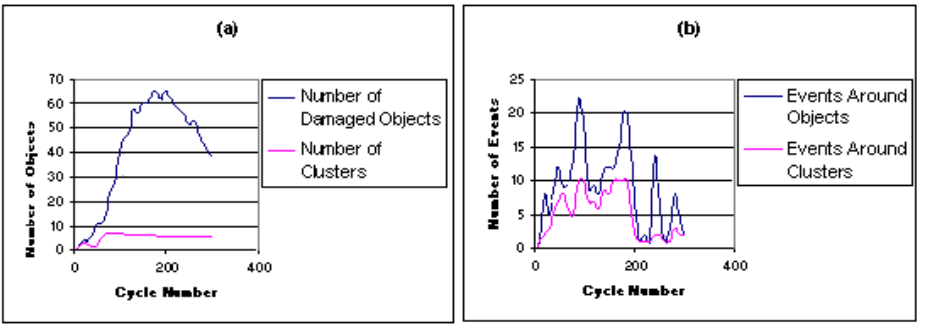


Fig. 3. Compactness of the Qualitative Representation vs. the Quantitative One. (a) The World size (b) The Events Rate

error (PEE) measures the inability of the representation to correctly capture all the objects assumed to be represented. The ***percent inclusion error (PIE)*** measures the inability of the representation to correctly exclude objects from the aspects of the world it captures.

$$PEE = \frac{damaged\,world\,objects\,not\,included\,in\,a\,cluster}{total\,number\,of\,world\,objects} \times 100\%$$
$$PIE = \frac{safe\,world\,objects\,included\,in\,a\,cluster\,yolk}{total\,number\,of\,world\,objects} \times 100\%$$

For our qualitative representation, *PEE* was 0.33% while the average PIE was 0.19%. Hence, the two error measures are practically non-existent.

6.3 Expressiveness

Measures the extent to which the representation is able to describe actions/plans.

We have formulated a sample query to assess expressiveness. The query reads: *List all fire brigade agents currently at the border of a fire.* As for the qualitative representation, the task is to extract the fire brigade agents internal to the white of a fire cluster. We obtain the set *at_border*.

$$\textbf{\textit{at\_border}} = [ag | \exists fc_i \in FClusters, Possible\_INTERNAL_v(fc_i, ag, t\text{-}x, t) = true].$$

For the quantitative representation, the task is comprised of finding the buildings that are not on fire whose neighbor is on fire and subsequently checking if a fire brigade is located at it or near it.

We have added a stopper flag that is turned on if the engine is unable to compute a result by the end of one simulation cycle and produces -1 for the number of agents satisfying the query if no result is produced.

The experiment results show that the qualitative representation reported 20% more agents satisfying the query than the quantitative one.

7 Conclusion and Future Work

By implementing a commentator that describes online developments in a dynamic urban disaster space, we were able to investigate the performance of a spatiotemporal representation for vague EggYolk regions. The performance criteria that we used try to measure the compactness, the faithfulness in capturing the underlying world correctly, and expressive power of the representation.

The commentator is the first step towards implementing a qualitative language to control the actions of the rescue agents as they perform their tasks. Our aim is to develop a language that allows us to express qualitative strategies like: 'if a fire is too big and there are not enough agents to put it out then leave it', or 'stay in a fire that is small until you put it out'. We believe that such an abstract language will offer significant advantages in a dynamic environment.

References

1. J. Allen. Maintaining knowledge about temporal intervals. Communications of the ACM. **26** (1983) 832-843.
2. N. Cohn and N. Gotts. The 'Egg-Yolk' representation of regions with indeterminate boundaries. GISDATA **2**(1996) 171-187.
3. A. Cohn and S. Hazarika Qualitative Spatial Representation and Reasoning: An Overview. Fundamenta Informaticae **43(1-2)** (2001) 2-32.
4. A. Cohn, and D. Magee and A. Galata. and Hogg, D. and Hazarika, S. Towards an Architecture for Cognitive Vision using Qualitative Spatio-Temporal Representations and Abduction. Spatial Cognition III (2003) 232-248.
5. J. De Kleer and J. Brown. A Qualitative Physics Based on Confluences. Artificial Intelligence **24(1-3)** (1984) 7-83.
6. M. Egenhofer. Query processing in spatial-query-by-sketch. Journal of Visual Languages and Computing **8** (1997) 403-424.
7. M. Escrig. Qualitative Spatial Reasoning: Theory and Practice, Application to Robot Navigation. (1998) IOS Press.
8. B. Faltings. A Symbolic Approach to qualitative kinematics. Artificial Intelligence **56(2-3)** (1992) 139-170.
9. Z. Ibrahim and A. Tawfik. Spatio-temporal Reasoning for Vague Regions. Canadian AI Conference (2004) 308-321.
10. B. Kuipers and Y-T Byun . A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. Journal of Robotics and Autonomous Systems **8** (1991) 47-63.
11. P. Muller. Space-Time as a primitive for space and motion. Proceedings of FOIS (1998).
12. P. Muller. Topological spatiotemporal reasoning and representation. Computational Intelligence **18(3)** (2002) 420-450.
13. A. Raffaetà, C. Renso. and F. Turini. Enhancing GISs for Spatio-Temporal Reasoning. ACM international symposium on Advances in geographic information systems(2002), 42-48.
14. D. Randell, Z. Cui and A. Cohn. A spatial logic based on regions and connection. KR & R (1992) 165-176.
15. P. Sablayrolles. The Semantics of Motion. EACL (1995) 281-283.

The Cruncher: Automatic Concept Formation Using Minimum Description Length

Marc Pickett and Tim Oates

University of Maryland, Baltimore County, USA
marc@coral.cs.umbc.edu

Abstract. We present The Cruncher, a simple representation framework and algorithm based on minimum description length for automatically forming an ontology of concepts from attribute-value data sets. Although unsupervised, when The Cruncher is applied to an animal data set, it produces a nearly zoologically accurate categorization. We demonstrate The Cruncher's utility for finding useful macro-actions in Reinforcement Learning, and for learning models from uninterpreted sensor data. We discuss advantages The Cruncher has over concept lattices and hierarchical clustering.

1 Introduction

Concept formation is a form of abstraction that allows for knowledge transfer, generalization, and compact representation. Concepts are useful for the creation of a generally intelligent autonomous agent. If an autonomous agent is experiencing a changing world, then nearly every experience it has will be unique in that it will have at least slight differences from other experiences. Concepts allow an agent to generalize experiences and other data. In some applications, the concepts that an agent uses are explicitly provided by a human programmer. A problem with this is that the agent encounters problems when it faces situations that the programmer had not anticipated. For this reason, it would be useful for the agent to automatically form concepts in an unsupervised setting. The agent should be able to depend as little as possible on representations tailored by humans, and therefore it should develop its own representations from raw uninterpreted data.

One purpose of concept formation (and abstraction in general) is to concisely characterize a set of data (Wolff[9]). With this view, one can use *minimum description length* (MDL) as a guiding principle for concept formation. We have developed an algorithm, called The Cruncher, which uses this principle to form an ontology of concepts from a collection of attribute sets. The Cruncher is general in the sense that no further knowledge of the attribute sets needs to be provided.

In the past, several other methods have been proposed for concept formation. Perhaps the most common form of unsupervised concept formation is clustering. For an overview of some of these algorithms, see Fasulo[4]. A drawback to much

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 282–289, 2005.

of this work is that each item belongs in only one cluster. The Cruncher uses multiple inheritance, which allows it to overcome obstacles faced by strictly hierarchical classification systems such as hierarchical clustering and decision trees. For example, in Figure 1, The Cruncher describes the Penguin (which has attributes in common with Birds and Fish) as both a "bird" and an "aquatic" creature. Hierarchical clustering would force the Penguin to be in only one of these classes.

There has also been work on Ontology Formation (for example, see Gruber[6]), but much of this work is aimed at knowledge engineering, where a human's help is required to assist the ontology formation. The Cruncher is completely unsupervised in contrast. The Cruncher also allows for exceptions, which further sets it apart from most of the work in this community. The Cruncher's exceptions allows for better compression, and, for the UCI Zoo Database, allows for classifications that correspond to the human-developed classification. For example, the Platypus is described as an egg laying mammal, even though mammals are defined as not laying eggs.

The field of Formal Concept Analysis provides methods for producing Concept Lattices, which form an ontology with multiple inheritance. The main aspects that set The Cruncher apart from this work are: first, due to the rigidity of Formal Concept Analysis, exceptions are not allowed as they are for The Cruncher, and second, MDL is not a driving factor in producing the concept lattices. As we demonstrate in Section 3, The Cruncher provides better compression than standard Concept Lattice layout algorithms, such as that described in Cole[3]. For an overview of the field of Formal Concept Analysis see Ganter and Wille[5].

The idea for The Cruncher grew out of "PolicyBlocks", an algorithm for finding useful macro-actions in Reinforcement Learning (Pickett and Barto[7]). The Cruncher extends PolicyBlocks by framing it in terms of ontology formation and MDL, and by adding exceptions and the ability to create multiple levels of concepts.

This paper is organized as follows: Section 2 provides a description of our representation framework and The Cruncher algorithm. Section 3 reports the results of applying The Cruncher to a variety of domains including the UCI Zoo Database and macro-action formation in Reinforcement Learning. Section 4 discusses strengths and weaknesses of The Cruncher in light of these experiments, and suggests future research directions to address The Cruncher's weaknesses.

2 The Cruncher

Given a collection of sets of attribute-value pairs, where each attribute's value is from a finite alphabet, The Cruncher produces a concept ontology which uses inheritance to compress the collection of attribute sets. One can "flatten" this ontology by computing the inheritance of every node in it, and this flattened ontology will contain the original collection of attribute sets. The Cruncher uses a greedy approach for reducing the description length of the ontology (which is

initially just the collection of attribute sets). The description length is defined as the number of links in the ontology, be they "is-a" or "has-a" links, where an "is-a" link designates that one node inherits from another, and a "has-a" link specifies an attribute that a node has and that attribute's value. (Whether the number of nodes was also included in the description length did not significantly affect our results partly because this number usually closely corresponds with the number of links.) The Cruncher generates candidate concepts by finding the "intersections" of subsets of the current items in the ontology. These candidates are then evaluated by determining the reduction in description length if each were to be adopted as concepts in the ontology. If no candidate reduces the description length, then The Cruncher halts. If a candidate is selected, then it is added to the ontology, and all other concepts in the ontology inherit from it if it reduces their description length. If there is a contradiction in the value assigned to an attribute by the nodes from which a concept inherits, that term is simply discarded. Furthermore, if a concept has an attribute, but the node from which it inherits has a different value for that attribute, then the concept states what its value is for that attribute. Thus, exceptions are allowed.

The runtime of this algorithm depends on whether one generates all possible candidate concepts, which, theoretically, can be exponential in the number of sets of original attribute pairs. In practice, one can successfully generate ontologies by randomly generating only a subset of these candidates, thus yielding a polynomial time algorithm. There is also an incremental version of this algorithm which works works by inserting one new concept at a time, and generating candidates by intersecting that node with each of the other concepts in the ontology. The top candidate is selected (or none if no candidate yields a decrease in description length), then this candidate is inserted into the ontology following the same procedure.

3 Experiments

To test the general applicability of The Cruncher, we chose a diverse set of domains to which we applied our algorithm. The UCI Zoo Database is useful for gaining insight about The Cruncher's ontologies. PolicyBlocks provides a non-bitvector domain for which there is a definite performance measure (cumulative reward). The Circles world demonstrates The Cruncher's utility for a basic sensor time-series domain. The Concept Lattice Comparison provides an example of a typical concept lattice ontology and The Cruncher's ontology on the same data set.

3.1 The UCI Zoo Database

We applied The Cruncher to the Zoo Database from the UCI Machine Learning Repository[1]. This data has 101 animals which are represented as bit vectors of length 16 (we changed the integer attribute "number of legs" to the binary "has legs" for consistency). Additionally, this dataset has a classification for each animal, which we discarded. The ontology that was created by The Cruncher

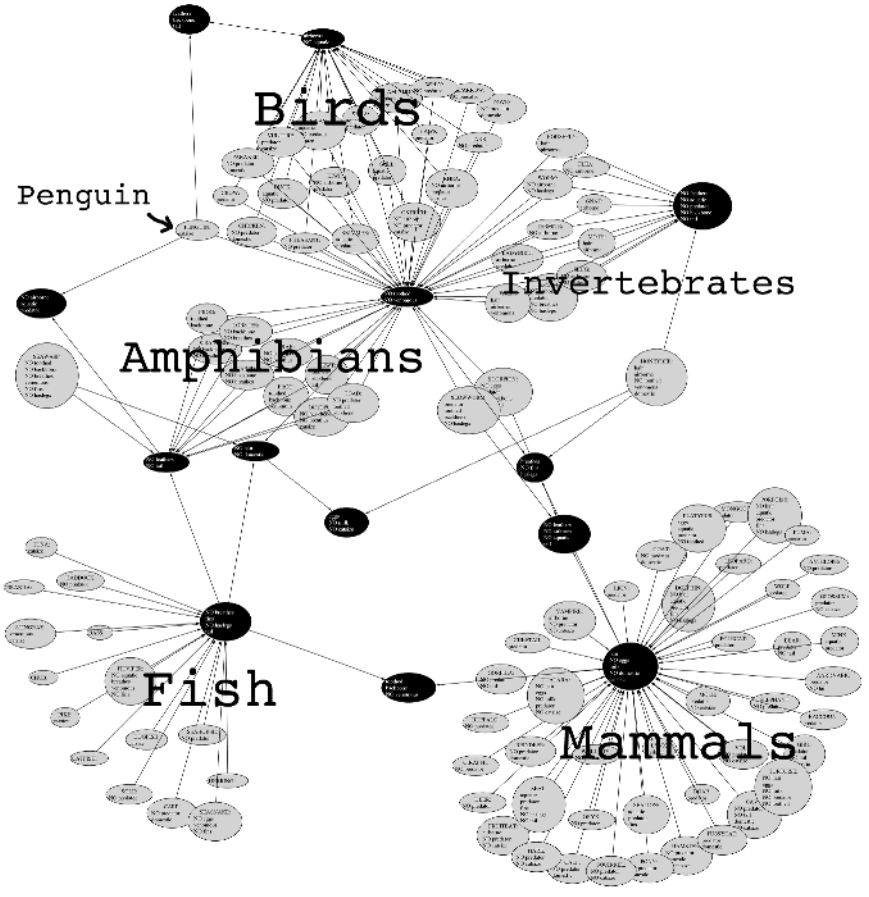


Fig. 1. The Automatically Created Zoo Ontology. Arrows represent inheritance, and the attributes and their values are listed in the nodes. Grey nodes are the original concepts in the database, and black nodes were created by The Cruncher. Mammals are grouped in the lower right, fish in the lower left, and the other three major clusters, from left to right, are, for the most part, reptiles/amphibians, birds, and invertebrates. Note, the multiply inheriting Penguin in the upper left. The class of birds is divided just for the Penguin, and the Penguin shares traits with the aquatic animals ("not airborne", "aquatic", "predator")

is shown in Figure 1. An interesting outcome is that the animals are arranged according to their classification even though this classification data was never provided. For example, there is almost a one to one correspondence to the animals that inherit from the black node in the lower right and the class Mammal. (The Tuatara and the Tortoise are the only exceptions.) Birds, Fish, and Invertebrates are likewise grouped together. The utility of allowing exceptions is demonstrated by the Platypus, which is classified as an egg-laying mammal (even though mammals are asserted as not laying eggs). The utility of having multiple

inheritance is demonstrated by the case of the Penguin, which shares traits with both aquatic life and birds.

3.2 PolicyBlocks: Creating Useful Macro-actions

In Pickett and Barto[7], it was demonstrated that for policies in a Markov Decision Process, certain concepts, which are effectively those created in the first level of abstraction in The Cruncher, can be used as useful macro-actions. These macro-actions outperform hand-chosen macro-actions such as getting to the "doorways" of the rooms in a grid-world. We started with a 20 by 20 grid-world structure (see Figure 2), and produced full policies leading to each of 20 randomly selected goal states. These policies are represented as a collection of 400 attribute-values, where the attributes are each of the 400 states, and the values are one of *up*, *down*, *left*, and *right*.

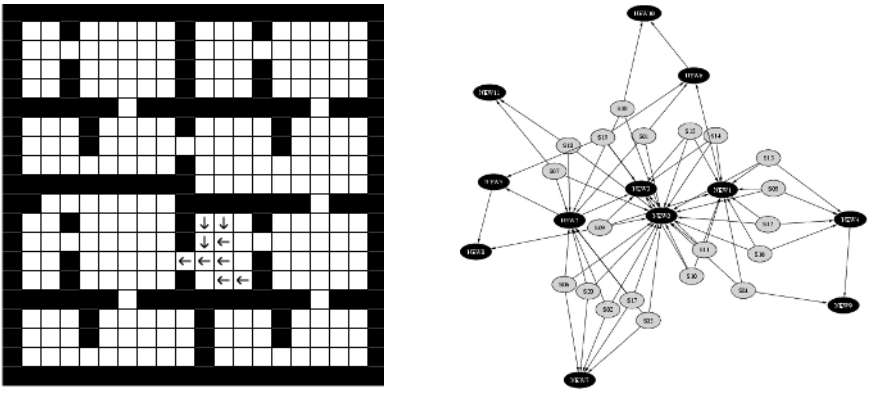


Fig. 2. A macro-action ontology. On the left is a macro-action generated by applying The Cruncher to a set of policies on a grid-world. The structure of the ontology is shown on the right. Each grey node corresponds to a full policy over the grid-world. Each black node is a sub-policy, or macro-action, that was produced by The Cruncher. For example, the macro-action encoded by the bottommost node in this ontology is that shown in the grid-world on the left. The arrows are "is-a" links, so every full policy can be thought of as the composition of all the sub-policies from which it inherits (in addition to the grey node's own modifications). Thus, each black node can be thought of as a "building block" for a full policy. (This is the origin of the term "PolicyBlocks")

3.3 The Circles World

We applied The Cruncher to the Circles domain. This is a simple 2 Dimensional physical simulation where the "particles" are circles that are "gravitationally" attracted to each other. The circles are on a torus-shaped world (i.e., the top and bottom "wrap around" to each other as do the left and right sides). There

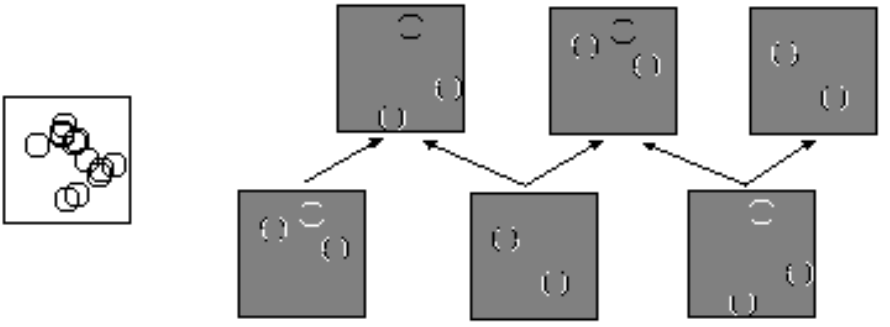


Fig. 3. The Circles World On the left is a snapshot of The Circles World, which is represented as a set of 2,500 (50 by 50) features corresponding to each of the 50 by 50 pixels. The 12 circles shown are in the process of orbiting each other in their gravitational dance. One the right is the circles top-level concept ontology. The grey areas are unspecified, and the lower 3 squares show only their *hasA* sets (as opposed to being flattened). The edges are "is-a" links. Thus, one can inherit from these "Circles concepts" to help compose a full Circles snapshot, such as the one shown on the left, just as one can use the "Policy blocks" in Figure 2 to compose full policies

are no collisions, but when the circles are sufficiently close to each other, their attraction is nullified until they are farther apart.

We provided The Cruncher with 50 "snapshots" from this simulation, where a snapshot is a set of 2,500 bits representing a 50 by 50 bitmap (see Figure 3). Note that the 2,500 bits are a raw data set. That is, no organization was provided to The Cruncher about whether, for example, Bit-1837 had anything more to do with Bit-1836 than it did with Bit-2354. This is fundamentally the same type of problem that Pierce and Kuipers[8] addressed using a different method based on statistical analysis. At the level of these bits, the concept of a circle is a fairly abstract entity. Here, The Cruncher has taken some steps toward describing the notion of a circle in that it has found that bits that form a circular pattern (when arranged in a bitmap) have something to do with each other. Figure 3 shows the top level concepts in the ontology created by The Cruncher for this domain. Noticing these correlations is the beginning of a theory of 2 Dimensional space, that is a similar result (though by different means) to the first step produced by Pierce and Kuipers[8].

3.4 Concept Lattice Comparison

The Cruncher yields better compression results than standard concept lattice layout algorithms. For example, the ontology produced by The Cruncher (Figure 4) for a Biological Organism domain had both fewer edges and fewer nodes (28 and 11, respectively) than that produced by Cole's method[3] (37 edges and 18 nodes). Additionally, formal concept lattices do not allow for exceptions, whose usefulness was shown in the case of the Platypus in the Zoo Database (see Figure 1).

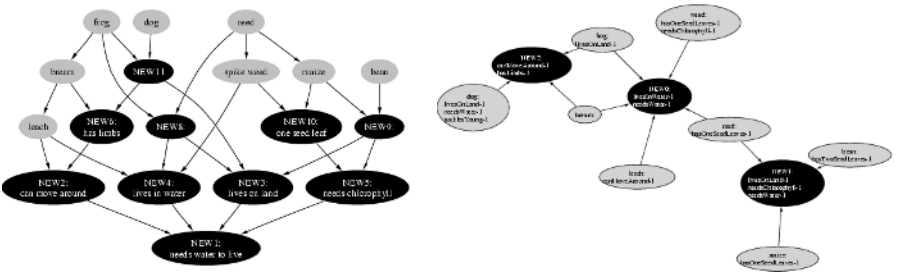


Fig. 4. Concept Lattices. A Biological Organism Domain adapted from Cole[3]. The ontology on the left was produced by a concept layout algorithm from that same paper. It has 37 edges (including the 9 "attribute" edges) and 18 nodes, which is more edges and more nodes than the ontology produced by The Cruncher shown on the right, which has 28 edges (including 17 "attribute" edges) and 11 nodes

4 Discussion

The strength of The Cruncher lies in its simplicity and its generality. The Cruncher was directly applied (i.e., with minimal massaging of the input representation) to diverse domains with positive results. Therefore, we believe that the basic ideas underlying The Cruncher may play a pivotal role in abstraction algorithms in general. The principle of MDL may be a part of a more general principle of Balance of Computational Resources. Occasionally, it is useful to cache the result of an inference, thereby trading memory for time. For example, one needs only Euclid's 5 postulates and an inference mechanism to produce all of Euclidean geometry. In practice, it's often useful to "cache" theorems rather than rederiving them even though this results in a larger description length. This resource balance may be viewed as finding Pareto optima in model space where models are evaluated by their time and memory requirements, and their accuracy. Alternatively, a model may be given a score based on some "exchange rate" among these resources. There have also been arguments that MDL alone might not be sufficient to produce useful concepts (Cohen et al.[2]) since compression tends to find frequent, but not necessarily meaningful results. However, sheer frequency is not the only factor in The Cruncher, and it would be interesting to apply our algorithm to the data set used by Cohen et al.[2] for which standard compression algorithms fail to produce meaningful results.

There are several extensions that can be made to The Cruncher. Among the most immediate of these is exploiting the heterarchy of the created ontology to speed up further crunching. This might be especially useful in the incremental version. There are some forms of concept formation that people do, but The Cruncher does not handle. For example, a person can watch a bird's eye view of a simulation of highway traffic, and quickly point out traffic jams. The person could tell you where the traffic jams are, how big they are, and how fast they are moving (traffic jams tend to move in the direction opposite that of the cars in them). A traffic jam is different from a cluster of cars because, like particles in

a wave, individual cars enter and exit a traffic jam, but the traffic jam remains. The circles in The Circles domain are like traffic jams in the sense that certain pixels turn on and off, but a pattern (i.e., the circle) remains consistent. The Cruncher also has no notion of precedence or dynamics. For example, the order of the snapshots of the circles domain was discarded. If one adds the ability to represent order, relationships, and dynamics as attribute-values, then The Cruncher can be used to organize and form concepts from stories, processes, and properties.

References

1. C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.
2. Paul R. Cohen, Brent Heeringa, and Niall Adams. `icdm2002.pdf`An Unsupervised Algorithm for Segmenting Categorical Time Series into Episodes. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002)*, pages 99–106, 2002.
3. Richard Cole. Automated layout of concept lattices using layered diagrams and additive diagrams. In *ACSC '01: Proceedings of the 24th Australasian conference on Computer science*, pages 47–53. IEEE Computer Society, 2001.
4. D. Fasulo. An analysis of recent work on clustering algorithms, 1999.
5. B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag, New York, 1999.
6. T. R. Gruber. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993. Kluwer Academic Publishers.
7. M. Pickett and A. Barto. Policyblocks: An algorithm for creating useful macro-actions in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2002.
8. David Pierce and Benjamin Kuipers. Map learning with uninterpreted sensors and effectors. *Artificial Intelligence*, 92:169–229, 1997.
9. J. G. Wolff. Information compression by multiple alignment, unification and search as a unifying principle in computing and cognition. *Artif. Intell. Rev.*, 19(3):193–230, 2003.

Experiments with Multiple Abstraction Heuristics in Symbolic Verification

Kairong Qian, Albert Nymeyer, and Steven Susanto

School of Computer Science & Engineering,
The University of New South Wales, Sydney Australia
{kairongq, anymeyer, ssus290}@cse.unsw.edu.au

Abstract. In this work we investigate a symbolic heuristic search algorithm in a model checker. The symbolic search algorithm is built on a system that manipulates binary decision diagrams (BDDs). We study the performance of the search algorithm in terms of the number of BDD operations, size of the BDDs, number of nodes they contain and runtime. We study the heuristic distribution of the state space, we measure effort by computing the *mean heuristic value*, and we compare single and multiple heuristics. In the case of multiple heuristics, we consider admissible and non-admissible merge strategies. We experiment on problems from a variety of domains. We find that multiple heuristics can perform significantly worse than single heuristics in symbolic search in at least one domain. In general, the effect of the heuristics on the symbolic search in the different domains varies markedly, and we conjecture that the different behaviour is caused by intrinsic differences in the characteristics of the state space.

1 Introduction

Formal verification techniques such as model checking [3] have gained much attention in the past decade. From the time Binary Decision Diagrams (BDDs) were introduced [2], symbolic model checking [16] that uses BDDs have been very successful in handling designs that have extremely large state spaces. While BDDs can represent the state space compactly, symbolic model checking of course still suffers the problem of "state space explosion" as it still must enumerate the full state space. This enumeration is typically done using a 'blind' breadth-first or depth-first search strategy. The blindness of the search is an unnecessary handicap that results in many irrelevant states being visited.

Heuristic-search algorithms such as A* and IDA* have been employed in AI research to solve many hard state-space search problems [13]. The big advantage of using a heuristic search strategy is that only part of the state space needs to be searched. Many verification of system design techniques, for example model checking, involve searches for a defect in a model. Coupling symbolic model checking with heuristic search techniques yields a more efficient technique to detect defects.

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 290–304, 2005.

In [7, 9, 12, 17] traditional explicit-based heuristic search algorithms have been modified to use BDDs to represent the state space. These methods can enhance the "bug-hunting" capabilities of symbolic model checkers because of the action of the heuristics in guiding the search. In [6], heuristics are classified as *property-specific*, *structural* and *abstraction heuristics*. Property-specific heuristics can usually be derived from analysing the property that is being verified. Structural heuristics guide the search algorithm by taking into account the structure of the state space. These two heuristics often work well in explicit state model checking. The third class of heuristics are developed from the abstractions of the model that is being verified. In this class of heuristics, the abstraction is a 'relaxation' of the system, and is generated by removing complicating detail from the concrete model. Abstractions here are called "patterns", and the resulting heuristics "pattern databases".

A pattern database [4] stores the distance of a pattern to some sub-goal state. Pattern databases were originally developed to solve many hard combinatorial puzzles in AI, e.g. n^2-1 puzzles. In [5, 19, 6] this notion is extended and combined with data abstraction, which is another technique to reduce the size of the state space. In earlier work [19, 20], we have seemlessly integrated symbolic pattern databases and symbolic model-checking algorithms.

Research in pattern databases has been well studied in AI. Holte and Hernádvölgyi [11] studied the trade-off between time and space for memory based heuristics. Korf and Felner [14] used so-called disjoint pattern databases, and solved very challenging search problems like Rubick's cube. Multiple pattern databases have been comprehensively investigated in [10], as well as the relationship between the distribution of heuristic values and the search effort. Felner et al in [8] studied the generation of admissible heuristics by partitioning the problem into disjoint sub-problems. They use both static and dynamic schemes of partitioning. All this work only considered explicit-state search algorithms of course, and were only concerned with admissible heuristics, and only experimented on classical AI problems.

In this paper, we conduct experiments in heuristic- and BDD-based symbolic search in model checking for models from various domains. We seek to understand the effect that symbolic state enumeration brings to heuristic search, particularly with respect to each of the domains. We do this for both single-heuristic and multiple-heuristic search strategies. In the next section we briefly describe the framework we use for generating heuristics for a model checker. In Section 3 we lay the formal groundwork for the work. In Sections 4 to 8 we describe a series of experiments. We do an overall evaluation and draw conclusions in Section 9.

2 Abstraction-Guided Symbolic Model Checking

The technique we use is called the abstraction-guided symbolic model checking framework. In our previous work [19, 18, 20], we have used a single abstraction to guide the BDD-based symbolic search. In this work we extend this approach and consider multiple abstractions. We briefly describe the general approach here and

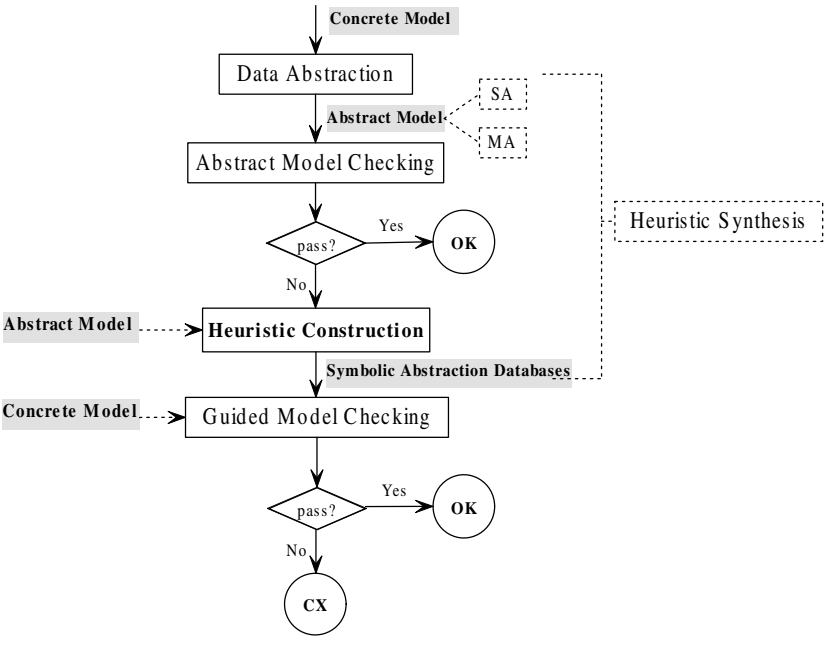


Fig. 1. The abstraction-guided model-checking framework

interested readers may refer to [19] or [20] for details. The abstraction-guided framework is depicted in Figure 1.

The process starts with the design, which we refer to as the concrete model. In the first step we generate a data abstraction of the concrete model. Note that in this step, we can generate more than one abstraction for the concrete model. We refer to a single abstraction as SA and multiple abstractions as MA in Figure 1. The abstract model(s) are taken as input by a symbolic model checker. If the model checker verifies the abstract model(s), we terminate as the data abstraction guarantees the soundness of the properties we are interested in. If the abstract model(s) fail the verification, we construct abstraction heuristic(s) using the abstract model(s). The guided model-checking algorithm is then invoked to check the concrete system using this heuristic as guide. The outcome of the heuristic model checker is either that the concrete model is verified, or a counter-example (*CX* in the figure) that will reveal the defect in the design (assuming the algorithm terminates of course).

Note that unlike other research in model checking, we use the same abstraction to (1) reduce the size of the model and (2) to guide the heuristic search algorithm. We have implemented this approach in a tool, called Golfer. This tool is built on top of the well-known symbolic model checker NuSMV[1].

[1] http://nusmv.irst.itc.it/

The heuristics that we construct from abstractions extend the notion of "pattern databases" developed in [4]. As our representation of the problem is based on BDDs, following [5, 6], we refer to these heuristics as *symbolic abstraction databases* (SADBs).

3 Symbolic Abstraction Databases

The terminology used in heuristic search in AI and in verification is quite different. In this section we define the notation we use, and explain our approach. We model AI search problems and the verification of safety properties using a finite-state model as follows.

Definition 1 (Finite Transition System). *A finite state transition system is a 4-tuple $M = (S, S_0, R, G)$, where*

- *S is a finite set of states*
- *$S_0 \subseteq S$ is a set of initial states*
- *$R \subseteq S \times S$ is a transition relation (or operator) that determines a set of successors for a given state $s \in S$*
- *$G \subseteq S$ is the set of goal states*

Definition 2 (Solution Path). *A path in a finite transition system (S, S_0, R, G), denoted by π, is a sequence of states $s_0, s_1, \ldots, s_n$ where $s_n \in G$ and for all $0 \leq i < n$, $s_i \in S \wedge (s_i, s_{i+1}) \in R$. If a path is a solution path then $s_0 \in S_0$. The length of π, written $|\pi|$, is just the number of states in the path.*

In verification, a solution path is called a *counter-example* as it demonstrates why the property that is being verified is not true.

Since we are only interested in symbolic heuristic search in this work, we encode M using Boolean expressions. Given a transition system $M = (S, S_0, R, G)$, we use a set of Boolean variables $X = \{x_1, x_2, \ldots, x_k\}$ to model the state space of M. A state can be represented by a truth assignment vector of X and all possible truth assignment vectors comprise the state space S. The Boolean functions $\mathcal{S}_0(x_1, x_2, \ldots, x_k)$ and $\mathcal{G}(x_1, x_2, \ldots, x_k)$ are characteristic functions that represent the states in S_0 and G (resp.). To encode R we need another set of Boolean variables $X' = (x'_1, x'_2, \ldots, x'_k)$ to represent the next state of a state s. Likewise, $\mathcal{R}(x_1, x_2, \ldots, x_k, x'_1, x'_2, \ldots, x'_k)$ is the characteristic function for R. In the discussion henceforth, we use $\mathcal{M} = (\mathcal{S}, \mathcal{S}_0, \mathcal{R}, \mathcal{G})$ to refer to the Boolean encoding of a transition system M.

If a system is modelled using a set of Boolean variables $X = \{x_1, x_2, \ldots, x_k\}$, we call $X_p \subset X$ a *pattern set*, and call those variables that are not in the pattern set $X_{\overline{p}}$. Given a pattern set and Boolean encoding of a transition system $\mathcal{M} = (\mathcal{S}, \mathcal{S}_0, \mathcal{R}, \mathcal{G})$, we can abstract the transition system as follows.

Definition 3 (Abstraction). *The abstraction of $\mathcal{M} = (\mathcal{S}, \mathcal{S}_0, \mathcal{R}, \mathcal{G})$ w.r.t a pattern set X_p is also a transition system $\hat{\mathcal{M}} = (\hat{\mathcal{S}}, \hat{\mathcal{S}}_0, \hat{\mathcal{R}}, \hat{\mathcal{G}})$ represented by its Boolean encodings, where*

- $\hat{\mathcal{S}}$ *is a disjunction of all minterms of variables in* X_p
- $\hat{\mathcal{S}}_0 \equiv \exists X_{\overline{p}} \mathcal{S}_0(x_1, x_2, \ldots, x_k)$
- $\hat{\mathcal{R}} \equiv \exists X_{\overline{p}} X'_{\overline{p}} \mathcal{R}(x_1, x_2, \ldots, x_k, x'_1, x'_2, \ldots, x'_k)$
- $\hat{\mathcal{G}} \equiv \exists X_{\overline{p}} \mathcal{G}(x_1, x_2, \ldots, x_k)$

Definition 4 (Symbolic Abstraction Databases). *Given a Boolean encoding* $\hat{\mathcal{M}} = (\hat{\mathcal{S}}, \hat{\mathcal{S}}_0, \hat{\mathcal{R}}, \hat{\mathcal{G}})$ *of a transition system, we call a set* $\sigma = \{(B_0, 0), (B_1, 1), \ldots, (B_n, n)\}$ *symbolic abstraction database such that:*

- (B_i, i)*, where* B_i *is a Boolean characteristic function and* $i \geq 0$
- $B_0 \equiv \hat{\mathcal{G}}$ *and* $B_i \equiv \exists X'_p (B_{i-1}[X_p / X'_p] \wedge \hat{\mathcal{R}})$ *for all* $0 < i \leq n$
- $B_n \wedge \hat{\mathcal{S}}_0 \not\equiv False$ *and* $B_i \wedge \hat{\mathcal{S}}_0 \equiv False$ *for all* $0 \leq i < n$
- $B_i \wedge B_l = \emptyset$ *for all* $i \neq j$

The length of a symbolic abstraction database is $|\sigma|$.

It is proved in [5, 19] if there is a solution path π in a concrete system M, there must exist a corresponding abstract solution path $\hat{\pi}$ in the abstraction $\hat{M}$ and $|\hat{\pi}| \leq |\pi|$. Note that the existence of $\hat{\pi}$ corresponding to π provides the theoretical justification for using $\hat{\pi}$ to guide the search. The lower-bound characteristic of the abstract solution path allows the symbolic abstraction database to be used as an admissible heuristic to estimate the actual number of transitions (distance) between the current and goal states in the concrete system.

Definition 5 (Disjoint SADBs). *Two symbolic abstraction databases* σ_1 *and* σ_2 *are disjoint if their corresponding pattern sets* X_{p1} *and* X_{p2} *are disjoint, i.e.* $X_{p1} \cap X_{p2} = \phi$.

Given a state s and characteristic function $\mathcal{F}_s$ in the concrete model and a symbolic abstraction database $\sigma = \{(B_0, 0), (B_1, 1), \ldots, (B_n, n)\}$, if $\mathcal{F}_s \wedge B_j \neq False$ for some pair (B_j, j), then we call the value j an *estimator* and denote it as $\sigma(s)$. Given a symbolic abstraction database, this estimator is unique as all B_i are disjoint.

Theorem 1 (Additiveness). *Let* $|\pi_s|$ *be the path length of* s *in* M*, and* σ_1 *and* σ_2 *be two disjoint symbolic abstraction databases. Let* $\sigma_1(s)$ *and* $\sigma_2(s)$ *be the estimator of* s *in each of them. Then* $\sigma_1(s) + \sigma_2(s) \leq |\pi_s|$.

This theorem guarantees that disjoint SADBs can be "added" together and the result will still be an admissible heuristic. In this work we in fact also consider pattern sets that are not disjoint. In verification, admissibility is less of an issue as we are more interested in finding just a good path to a defect in the system, not necessarily the shortest path. Our formal discussion about the disjoint SADBs actually shares many aspects with the disjoint pattern database heuristics in AI research [5, 14, 8].

We not only treat the system symbolically in this work, we also encode the guided heuristic search symbolically. In [19, 20] we describe how SADBs are

used by a symbolic model checker to detect safety violations. We further generalise the algorithm to use multiple SADBs. The search algorithm still works as in [19]; the only difference being the way SADBs are queried. The essence of the guided symbolic model checking algorithm is that each frontier BDD is split into several smaller, so-called sub-BDDs (representing *sub-frontiers*), where each of these sub-BDDs corresponds to a different heuristic value. Note that this splitting in necessary in the A* algorithm as the heuristics help drive the search. Edelkamp [7], who also studied the symbolic A* algorithm, used a different method however. In his method, the heuristic values are encoded into the BDD directly. It is not clear whether this method has any advantages over our method however. The heuristics in our research therefore fulfil two tasks: as a search guide and as a mechanism to split BDDs. The splitting is carried by the *restrict* operation (denoted as $\downarrow$) on BDDs [2].

Let $\mathcal{D}$ be a BDD representing a set of states of M and $\Phi = \{\sigma_1, \ldots, \sigma_m\}$ be a set of SADBs. The algorithm below splits the BDD representing $\mathcal{D}$ and assigns each sub-BDD an estimator according to the merge strategy of the SADBs.

Procedure Splitting $(\mathcal{D}, \Phi, m\_strategy)$
1 $result \leftarrow \{(\mathcal{D}, 0)\}$
2 **for** i in $1..m$ **do**
3 $temp \leftarrow result$
4 $result \leftarrow \{\}$
5 **for** each $(d, h) \in temp$ **do**
6 **for** each $(B_j, j) \in \sigma_i$ **do**
7 $I \leftarrow d \downarrow B_j$
8 $d \leftarrow d \wedge \bar{I}$
9 **if** $(I \neq \phi$ & $m\_strategy = add)$
10 $result \leftarrow result \cup \{(I, j + h)\}$
11 **if** $(I \neq \phi$ & $m\_strategy = max)$
12 $result \leftarrow result \cup \{(I, max(j, h))\}$
13 **if** $(d \neq \phi)$
14 $result \leftarrow result \cup \{(d, |\Phi|)\}$
15 **return** $result$

The input *m_strategy* is the merge strategy used on the SADBs in the symbolic heuristic search. The only possible values are *add* and *max*. Note that for disjoint SADBs, both are admissible. If Φ is not disjoint, only *max* will be admissible.

While abstraction-based heuristics can reduce the search space in both the AI and verification domains, the method used to derive abstractions in these domains is very different. For example, puzzles and planning problems in the AI domain can usually be physically modelled, so the abstraction of the problem often involves the detection of physical patterns. In the n^2-1 puzzle, for example, we have *corner* and *fringe* patterns [4]. In verification, however, problems are generally modelled by a large number of variables and the physical relationship between the variables is neither physical nor obvious, often due to the high level of concurrency of the model. Finding patterns in verification can be very difficult indeed.

In [20] we presented a procedure that automatically finds abstraction patterns by using a data dependency analysis. The idea is based on the notion that variables that are only indirectly related have a weaker influence on each other than variables that are directly related. We build a variable dependency tree rooted by the variables that occur in the goal state. The weakest variables, which appear furtherest away from the root in the variable dependency graph, are ignored. Note that this method does not always work well, particularly in AI problems, where typically all variables are directly dependent on each other.

4 Experiment Set-Up

In explicit-state heuristic search, the number of states (or nodes) that are generated by the search algorithm can be used to evaluate the effectiveness of the heuristic. In BDD-based heuristic search, however, we cannot use the number of states that are generated by the algorithm as states are symbolically represented by Boolean functions. We note that the number of nodes in a BDD is not related to the number of states it represents. In fact, the effort that a symbolic search algorithm must make to solve a problem is largely determined by the internal operations of the BDD engine, not the number of states in the system.

The following attributes will be used to capture the search effort in our experiments.

IM. The number of BDD image computations. These computations determine the successor states of the search. It is called the relational product and involves quantifier elimination, which is an expensive computation in symbolic model checking. (IM is related to the size of the *closed* set in the explicit-state A* algorithm.)

SP. The number of splitting operations. The splitting operation involves the 'restrict' operation on BDDs. It is also expensive. (SP is related to the size of the *open* set in explicit-state A* algorithm.)

ND. The total number of BDD nodes allocated. While the number of BDD nodes is not directly related to the number of states, it still reflects the memory usage of the algorithm and hence is the major memory measurement.

AS. The average size of all BDDs. Reducing the size of BDDs is important because some BDD operations have exponential time complexity in terms of the BDD size.

TM. The CPU time consumed by the search algorithm. In general the CPU time is strongly related to the values of IM and SP.

We have used 5 models in our experiments: two of them are puzzles from the AI heuristic search domain, and the other three are real-world design models of concurrent systems. Note that all models have at least one goal state that is reachable from the initial state set.

Name	Description	Type
puz	$n^2 - 1$ sliding tile puzzle ($N = 8$)	puzzle
perm	N-pancake puzzle ($N = 10$)	puzzle
dme	distributed mutual exclusive ring	circuit
ns	Needham-Schroeder public key protocol	protocol
peter	Peterson's mutual exclusion algorithm	protocol

To construct SADBs we need to define the pattern set that is to be used for the abstraction. For each puzzle model we use 4 different pattern sets that are commonly used in AI heuristic search literature. For each of the verification models, we use our data dependency analysis to also generate 4 pattern sets. The pattern sets for each model are not necessarily disjoint as the optimality of the solution path is not a primary concern in our work.

We ran the model checker for each of the (single) SADBs generated by the pattern sets. We also used multiple SADBs that were constructed by merging 3 of the 4 SADBs for each model. We in fact constructed multiple SADBs by all of the $\mathbf{C}_4^3$ combinations of SADBs and reported the best performance. As well, both the *add* and *max* merge strategies were studied.

Caveat: At this point, we should point out that symbolic search algorithms are not really suitable for solving the puzzle-like problems that often occur in AI heuristic search, where good heuristics are known. The advantage of the BDDs in manipulating sets of states in single operations is often outweighed by the computational complexity of these BDD operations [17]. In essence, only when the BDDs represent large sets of states does their use pay off. In general, for these types of problems, explicit-state searches are often faster and require less memory. We use these models in this work, however, for comparison purposes.

5 Heuristic Distribution Experiment

The aim of this experiment was to study the distribution of the heuristic over the state space. In Figure 2, we show the number of states in the SADB for different heuristic values. In our symbolic approach, all states with the same heuristic value is represented by a single BDD. To compute these results, we needed to calculate how many (abstract) states a BDD can represent (which is of course different to the number of nodes in the BDD). Note the logarithm scale on the axis for the number of states. The diagram on the left is taken from randomly chosen SADBs for the 8-puzzle model, and on the right, for the DME circuit model.

In the figure we observe that different abstraction show similar behaviour for each model. Comparing AI and verification, however, we observe that the number of states *increases* exponentially as the heuristic value increases in the case of the 8-puzzle, but *decreases* exponentially for the DME model.

We conjecture that this phenomenon is caused by fundamental differences in the nature of the state spaces in these domains. Puzzles, for example, often have few goal states, whereas safety properties in verification can be violated

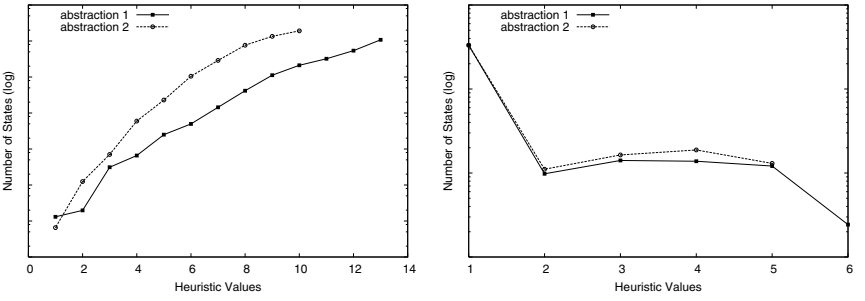


Fig. 2. 8-puzzle (left) and DME (right)

in many states, and hence there are many goal states. Having many goal states means that the value of the heuristic tends to remain small. When we construct SADBs, we use a backward breath-first traversal of the abstract model's state space. In the case of puzzles, the resulting search tree grows exponentially. In verification, the size of the search tree decreases as many states share the same predecessor. A conclusion one could draw from the behaviour we observe is that heuristic search using SADBs has less to gain in verification models than in AI models (in other words, the improvement over blind search will be less in verification).

6 Mean Heuristic Value Experiment

In this experiment we wished to study whether or not the *mean heuristic value* [15, 5], is a good predictor of the effort needed by the symbolic heuristic search. In explicit-state heuristic search, it is well known to be a good predictor of the search effort. The intuition is that it may not be in a symbolic setting because it does not factor in the added computational overhead that BDDs have.

The mean heuristic value (MHV) $\overline{h}$ of a SADB σ determines the overall distribution of heuristic values and is defined as follows.

$$\overline{h} = \sum_{i=1}^{|\sigma|} i \times (|\{s \in \sigma | \sigma(s) = i\}|/|\{s | s \in \sigma\}|) \quad (1)$$

Equation 1 actually computes the weighted mean heuristic value of σ. A high value for $\overline{h}$ indicates that a larger proportion of states have large heuristic values than have low values, and therefore the level of "informedness" is high. Ideally, the value of $\overline{h}$ for a SADB σ should be as close as possible to $|\sigma|$, and in that case it is said to be "well-informed".

We chose three models, and for each model we use 4 different SADBs. The results are shown below.

SADBs	MHV	IM	SP	ND	AS	TM
puz-1	**11.90**	**144**	**305**	**268421**	**361**	**1.050**
puz-2	12.30	679	1581	547046	404	3.390
puz-3	9.03	495	1043	453495	455	2.970
puz-4	8.30	1353	2769	400189	429	6.900
dme-1	2.73	616	1121	193004	2921	74.110
dme-2	**3.72**	**26**	**26**	**280683**	**3232**	**3.220**
dme-3	3.68	26	26	289808	6268	11.090
dme-4	3.57	26	26	524103	12394	45.540
ns-1	**8.13**	**28**	**106**	**92126**	**815**	**6.840**
ns-2	5.37	421	1693	371040	964	122.420
ns-3	6.16	1110	6361	629380	1843	389.670
ns-4	5.34	475	2073	519868	1874	228.980

The statistics that concern the BDDs come from the BDD engine of the model checker. Other statistics are generated from profilers. For each of the three models we give the results for each of the 4 SADBs. The value of MHV is calculated using equation 1. For each model, we bold the row that has the shortest run-time (TM).

For the puzzle model, the bolded row has the best performance in terms of every attribute. While the MHV for *puz-1* is not the highest one, the MHV is nevertheless a good predictor of performance.

For the two verification models, the SADB with the shortest run-time has the highest value of MHV. It is hence a very good predictor in the symbolic setting, contrary to our intuition.

Unrelated to the MHV, we observe that *dme-2*, *dme-3* and *dme-4* use exactly the same number of image computations and splitting operations (IM and SP). However, in spite of the fact that *dme-2* uses more BDD nodes (ND) than *dme-1*, and will hence use more memory, it is still the best performer.

7 Multiple SADB Experiment

This experiment concerns the main focus of this work, and that is compare the performance of single and multiple SADBs. Holte et al. [10] found that heuristic search that is based on multiple SADBs out-performs search based on single SADBs (for the same amount of memory). But Holte et al's work is based on an explicit-state search. So does it apply to a symbolic heuristic search as well?

We run each model with each of the 4 single SADBs. From these 4 SADBs, we created 10 multiple SADBs ($\mathbf{C}_4^2$ from combinations of 2 SADBs plus $\mathbf{C}_4^3$ from 3 SADBs). We ran the symbolic search on each model with each of these 10 heuristics. We show the results for just two of the models in the following table.

SADB(s)	IM	SP	ND	AD	TM
puz-sgl	144	305	268421	361	1.050
puz-mpl-1	133	284	353075	341	1.360
puz-mpl-2	203	440	409425	343	1.650
puz-mpl-3	160	349	299479	361	1.200
peter-sgl	40	97	109795	1332	4.414
peter-mpl-1	230	591	379365	816	14.273
peter-mpl-2	40	105	134202	822	16.442
peter-mpl-3	40	100	126754	716	16.357

In the table, the results for the single SADB (denoted by the *.sgl* suffix) are the ones that had the best performance. For the 10 multiple SADBs, we show just the best performing 3, and these are indicated by the *.mpl* suffix. We used both the *add* and *max* strategies to merge the SADBs.

Contrary to Holte et al's findings, the results in the table above show that multiple heuristics perform worse than single heuristics. The first two rows in the table, *puz-sgl* and *puz-mpl-1*, are particularly interesting as they reveal that even when the multiple SADB search uses less image computations and splitting operations (which we noted earlier are the primary determinants of the computational complexity), it performs worse than the single SADB search. We note the multiple SADB models use more BDDs nodes, and the average size of the BDDs is smaller, in both models.

These results are quite surprising. Multiple pattern databases in explicit-state heuristic search are effective because they improve the overall heuristic distribution and hence result in smaller search trees. In symbolic heuristic search, the heuristic is (also) used to split the frontier BDDs, and one conjectures, it is this computation that causes the problem. Thus, much of the effort of symbolic heuristic search is spent on splitting the BDDs, offsetting any gains that may be had from the higher-quality, multiple heuristic.

8 Merge Strategy Experiment

In our final experiment we compared the performance of the *add* and *max* merge strategies for multiple SADBs. We carried out this experiment by using both strategies to merge both 3 and 4 single SADBs into a multiple SADB. We note that the *add* strategy is not admissible so it can generate non-optimal paths.

Unlike our earlier experiments, this time we present tables for each of the problem domains separately. The domains are AI puzzles, electronic circuits and communication/security protocols. We do this because we found that the choice of merge strategy effected the performance in a different way for each of these domains. The only change in the table format to our earlier experiments is the addition of the attribute LE, which indicates the solution length returned by the search. You can see from this column when a search generated a non-optimal path.

AI Puzzles. In the table below we see the results for the AI puzzles.

SADBs	IM	SP	ND	TM	LE
puz-3-add	133	284	368552	1.450	24
puz-3-max	203	440	409425	1.650	24
puz-4-add	168	357	353075	1.360	24
puz-4-max	169	391	433784	1.880	24
perm-3-add	27	106	93224	5.720	13
perm-3-max	769	3446	1199309	294.340	11
perm-4-add	289	1833	237584	39.510	15
perm-4-max	2619	14427	859840	511.550	11

We observe that, while the *add* merge strategy can result in a non-optimal path, the resulting search is faster than that produced by the (optimal) *max* merge strategy. In fact, in the case of *perm*, it is one or two orders of magnitude faster. Note that there is almost the same difference between the *max* and *add* strategies in the number of image computations and partition operations, so the result is not surprising. There is a trade-off here: speed comes at the cost of optimality.

Electronic Circuit. This model has been constructed from a real electronic circuit design and has been a widely used benchmark for symbolic model checking.

SADBs	IM	SP	ND	TM	LE
dme-3-add	169	232	142574	10.520	37
dme-3-max	616	1122	186500	77.380	27
dme-4-add	169	232	171070	10.260	41
dme-4-max	326	546	198282	16.220	27

We observe that the *add* strategy clearly results in a faster model checker than *max* but at the cost of a much longer path to a goal state.

Communication/Security Protocols. The two communication protocol models generate quite different results.

SADBs	IM	SP	ND	TM	LE
ns-3-add	1452	11831	218287	187.460	19
ns-3-max	32	273	113118	7.040	14
ns-4-add	1954	14800	318377	228.680	19
ns-4-max	130	1054	130216	12.580	14
peter-3-add	230	591	379365	14.273	49
peter-3-max	40	100	130955	2.970	41
peter-4-add	79	199	151200	25.890	49
peter-4-max	40	105	134202	16.442	41

Quite the opposite of the previous results, the *add* strategy for these models results in a model checker that takes a lot longer to find a longer, non-optimal path to a goal state. Clearly an unsatisfactory heuristic for this class of model.

In summary, the inadmissibility of the *add* merge strategy may lead to (very) sub-optimal paths, and a substantial speed-up in the search in some models, but a worsening in others.

9 Evaluation and Conclusion

Predicting how and when BDD-based heuristic search algorithms will perform better than explicit-state algorithms is extremely difficult. It is well known, for example, that finding an optimal variable ordering for BDDs is an NP-hard problem [1]. We have not considered the variable ordering in this work yet (but have in earlier work [20]). BDDs can be 'exponentially' efficient in representing very large sets of states, and because of this, can be vastly superior to explicit-state search algorithms. However, when the sizes of the sets they represent are not large, the

computational overhead of manipulating BDDs can result in very poor performance indeed. The problem of predicting performance is compounded when you add heuristics, and compounded again when you allow multiple heuristics. So the problem we are addressing is indeed very difficult.

In AI, finding the shortest path to the goal state is paramount. In verification, finding a 'reasonably short' path is often sufficient. More important is the time it takes to find this path. The reason for this is that the model checker is being used as a debugger, and hence we need to know quickly whether there is an error in the specification or not. In verification therefore, we are often prepared to sacrifice optimality for speed.

While we have tried to be comprehensive in the experiments, we do of course:

- have only a small sample of models,
- have just a few abstractions (derived automatically for the verification models)
- have just 2 merge strategies: one admissible, one non-admissible.

On the positive side, we have attempted to bridge disparate fields, AI and verification, by understanding the behaviour of a technology, symbolic heuristic search, that is common to both. We can summarise the results of our experiments in the following way:

- The distribution of the heuristic over the state space is different for AI models than verification models. This difference could be caused by different characteristics of the state space: for example, there are typically more goal states in verification than in AI problems, and verification state spaces are less tree-like.
- The MHV still makes a good predictor of effort in symbolic heuristic search.
- Contrary to Holte et al. [10], we found that multiple symbolic heuristics performed worse than single symbolic heuristics. We conjecture that this is caused by the overhead of splitting the BDDs. Note that in some cases splitting a BDD results in larger BDDs than the original. This is an unfortunate side-effect of this method that cannot easily be avoided.
- If you have a naturally good heuristic distribution, as AI problems tend to have, then an 'aggressive', non-optimal merge strategy like *add* will result in multiple SADBs that perform much better than single SADBs; albeit at the possible cost of optimality.
- Verification problems that have poorly, or narrowly distributed heuristics should not use non-optimal merge strategies.

AI puzzles and electronic circuits typically have very dense state spaces, while protocol models have relatively sparse state spaces. Intuitively, dense state spaces will contain a larger number of solution paths than sparse state spaces. This could be the cause of the behaviour we observe in the merge-strategy experiment. A non-optimal strategy like *add* enables heuristic search algorithms like A* and IDA* to guide aggressively during the search because it increases the proportion of states that have larger heuristic values, and penetrates deeply into the state

space. Consequently, however, the search may miss shallow solutions and fruitlessly pursue dead-end paths. Note that AI puzzles and circuits have relatively fewer goal states than protocol models. We conjectured in Section 5 that this was the cause of the behaviour that we observed in the heuristic-distribution experiment. The topology of the state space is therefore potentially very important in determining the performance of the symbolic search.

The future work we are planning is the following:

- Take the BDD variable ordering into account.
- More work needs to be done to determine how to abstract the system automatically. This is of course an open research question. Currently our approach using a data dependency analysis is simplistic.
- To restrict the sizes of BDDs, we need to consider more effective mechanisms such as "high density" reachability analyses [21]. This is especially important for splitting the frontier BDDs,
- We need to understand which characteristics of the state space are important for the performance of the guided and symbolic approach. While we have tried to do this by considering models from different domains, more focussed experiments that shed light on this issue are needed. It would appear that you need to know what the topology of the state space is before deciding which search algorithm to apply.

References

1. B. Bollig and I. Wegener. Improving the variable ordering of obdds is np-complete. *IEEE Trans. Computers*, 45(9):993–1002, 1996.
2. R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transaction*, C-35(8):677–691, Aug 1986.
3. E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
4. J. C. Culberson and J. Schaeffer. Searching with pattern databases. In *Proc. of the 11th Biennial Conf. of the Canadian Society for Computational Studies of Intelligence, Toronto, Ontario, Canada, May 21-24*, volume 1081 of *LNCS*, pages 402–416. Springer-Verlag, 1996.
5. S. Edelkamp. Symbolic pattern databases in heuristic search planning. In *Proc. of the Sixth Int. Conf. on Artificial Intelligence Planning Systems, April 23-27, Toulouse, France*, pages 274–283. AAAI, 2002.
6. S. Edelkamp and A. Lluch-Lafuente. Abstraction databases in theory and model checking practice. In *Proc. of Workshop on Connecting Planning Theory with Practice, Int. Conf. on Automated Planning and Scheduling, ICAPS, Whistler, Canada*, 2004.
7. S. Edelkamp and F. Reffel. OBDDs in heuristic search. In *KI-98: Advances in Artificial Intelligence, 22nd Annual German Conf. on Artificial Intelligence, Bremen, Germany, September 15-17, Proc.*, volume 1504 of *LNCS*, pages 81–92. Springer-Verlag, 1998.
8. A. Felner, R. E. Korf, and S. Hanan. Additive pattern database heuristics. *J. Artif. Intell. Res. (JAIR)*, 22:279–318, 2004.

9. E. Hansen, R. Zhou, and Z. Feng. Symbolic heuristic search using decision diagrams. In *Proc. of the Symp. on Abstraction, Reformulation and Approximation, Alberta, Canada*, volume 2371 of *LNCS*, pages 83–98. Springer-Verlag, 2002.
10. R. Holte, J. Newton, A. Felner, R. Meshulam, and D. Furcy. Multiple pattern databases. In *Proc. of 14th Intl. Conf. on Automated Planning & Scheduling (ICAPS)*, pages 122–131. AAAI, 2004.
11. R. C. Holte and I. T. Hernádvölgyi. A space-time tradeoff for memory-based heuristics. In *AAAI/IAAI*, pages 704–709, 1999.
12. R. M. Jensen, R. E. Bryant, and M. M. Veloso. Seta*: An efficient bdd-based heuristic search algorithm. In *Proc. of the Eighteenth National Conf. on Artificial Intelligence and 14th Conf. on Innovative Applications of Artificial Intelligence, Alberta, Canada*, pages 668–673. AAAI Press, 2002.
13. R. Korf. Finding optimal solutions to to Rubik's cube using pattern databases. In *Proc. of the 14th National Conf. on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conf., July 27-31, Providence, Rhode Island*, pages 700–705. AAAI Press/The MIT Press, 1997.
14. R. E. Korf and A. Felner. Disjoint pattern database heuristics. *Artif. Intell.*, 134(1-2):9–22, 2002.
15. R. E. Korf, M. Reid, and S. Edelkamp. Time complexity of iterative-deepening-a*. *Artif. Intell.*, 129(1-2):199–218, 2001.
16. K. McMillan. *Symbolic model checking.* Kluwer Academic Publishers, Boston, MA, 1993.
17. A. Nymeyer and K. Qian. Heuristic search algorithm based on symbolic data structures. In *Proc. of the 16th Australian Joint Conf. in Artificial Intelligence, Perth, Australia, 3-5 December*, volume 2903 of *LNAI*, pages 966–979. Springer-Verlag, 2003.
18. K. Qian and A. Nymeyer. Abstraction-based model checking using heuristical refinement. In *Proc. of the 2nd Int. Symp. on Automated Technology for Verification and Analysis, ATVA'04*, pages 165–178. Springer-Verlag, 2004.
19. K. Qian and A. Nymeyer. Guided invariant model checking based on abstraction and symbolic pattern databases. In *Proc. of the 10th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, Barcelona, Spain,*, volume 2988 of *LNCS*, pages 497–511. Springer-Verlag, 2004.
20. K. Qian and A. Nymeyer. Abstraction-guided model checking using symbolic IDA* and heuristic synthesis. In *Submitted to FORTE'05 for publication*, 2005.
21. K. Ravi and F. Somenzi. High-density reachability analysis. In *ICCAD '95: Proc. of the 1995 IEEE/ACM international conference on Computer-aided design*, pages 154–158. IEEE Computer Society, 1995.

Probabilistic Abstraction of Uncertain Temporal Data for Multiple Subjects

Michael Ramati and Yuval Shahar

Medical Informatics Research Center, Department of Information Systems Engineering,
Ben-Gurion University, Beer-Sheva 84105, Israel
{ramatim, yshahar}@bgu.ac.il

Abstract. Several Systems have been designed to solve the task of abstraction of time-stamped raw data into domain-specific meaningful concepts and patterns. All approaches had to some degree severe limitations in their treatment of incompleteness and uncertainty that typically underlie the raw data, on which the temporal reasoning is performed, and have generally narrowed their interest to a single subject. We have designed a new probability-oriented methodology to overcome these conceptual limitations. The new method includes also a practical parallel computational model that is geared specifically for implementing our probabilistic approach.

1 Introduction

The commonly occurring task of Temporal Abstraction (TA) was originally defined as the problem of converting a series of time-oriented raw data (e.g., a time-stamped series of chemotherapy-administration events and various hematological laboratory tests) into interval-based higher-level concepts (e.g., a pattern of bone-marrow toxicity grades specific to a particular chemotherapy-related context) [1]. Former solutions [1-4] although being evaluated as fruitful, maintained several unsolved subproblems. These subproblems seem common to some of other methods suggested for solving the TA task, such as [5-7]. Thus, considering these challenging subproblems suggests an additional method.

At least three subproblems in the former methods can be pointed out, which we propose to solve through the method discussed in this paper. First, raw data, to which the temporal reasoning is being applied, are assumed as certain – that is, typically no mechanism is suggested for handling the inherent impreciseness of the tests taken to obtain the data. Second, current mechanisms used for completing missing time-oriented data are typically not sound and are incomplete. For example, in the case of the KBTA method, a knowledge-based interpolation mechanism is used [8]. However, completion of missing values is supported only for bridging gaps between two intervals, in which the proposition (e.g., anemia level) had the same value (e.g., moderate anemia). Furthermore, the value concluded by inference is too crisp, and a threshold is used for computing it with absolute certainty, eliminating uncertainty and leading to potentially unsound conclusions. Third, no special mechanism has been devised for multiple subject abstraction. That is, so far temporal abstraction was performed on a single subject only.

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 305 – 312, 2005.

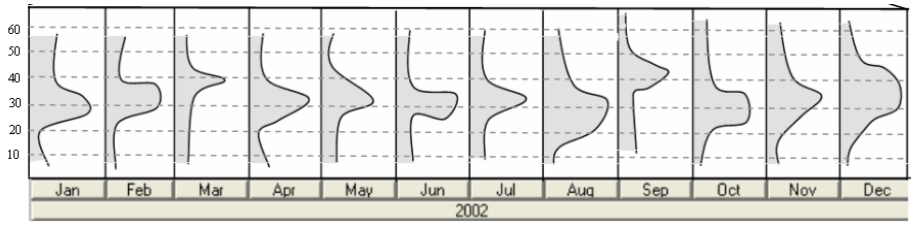


Fig. 1. A typical instance of using the PTA method: the value (*vertical axis*) distribution of a certain concept appears for different (in this case consecutive) periods along the time axis. The concept, which can be either raw or abstract, and the specification of the set of periods (including the time granularity) are determined by the application using the PTA method

The proposed method, *Probabilistic Temporal Abstraction* (PTA), decomposes the temporal abstraction task into three subtasks, that solve the case of a single subject, and two more subtasks that solve the case of multiple subjects. In addition to overcoming the above mentioned subproblems, we also propose a design for a parallel computational model that implements the method.

2 The Subtasks of the PTA Method

Several basic notions in probability theory relate to time, and are important when considering a probabilistic temporal model, task or mechanism. A *stochastic process* {X(t): t in T} is a set of random variables. The index is often interpreted as time, and thus X(t) is referred as the *state* of the process at time t. The set T is called the *index set* of the process. The subtasks specified below are defined in terms of these notions.

Single-Subject Subtasks. Temporal abstraction for a single subject requires one basic subtask, interpolation, and two interpolation-dependent subtasks – coarsening and transformation:

Temporal Interpolation. Estimating the distribution of a stochastic process state, given the distributions of some of its other states (Fig. 2). For example, estimating the distribution of raw hematological data or derived concepts (such as bone-marrow toxicity grades) during a week in which raw data were not measured, using the distribution of values before and after that week. Applying the interpolation subtask does not increase the abstraction level of the underlying stochastic process, but rather serves the role of a core operation that enables the application of actual temporal abstraction.

Temporal Coarsening. Applying an aggregation function to a stochastic subprocess (Fig. 3). The coarsening subtask abstracts over the time axis and is aimed at the calculation of a stochastic process at a coarser time granularity.

Temporal Transformation. Generating a stochastic process, given stochastic processes of a lower abstraction level. For example, deriving bone-marrow toxicity grade distribution, given the distributions of the raw white blood cell and platelet counts. The transformation subtask abstracts along the (concept) abstraction-level axis.

Multiple-Subject Subtasks. Applying the TA task to multiple subjects requires extra subtasks, such as the ones explicated below. However, these subtasks fit also sophisticated needs of abstraction for a single subject.

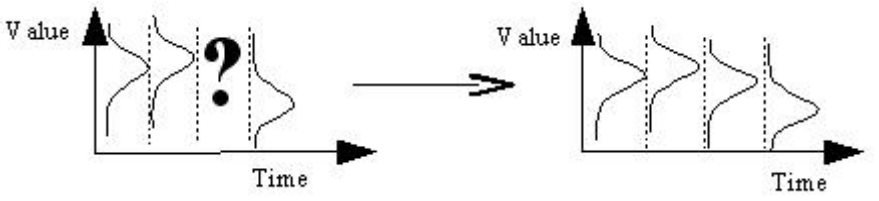


Fig. 2. An illustration of the interpolation subtask. Given the value distribution at several time points, there is a need to calculate an unobserved value distribution. The solution suggested by the PTA considers all value distributions

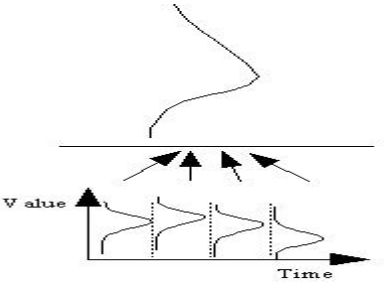


Fig. 3. An illustration of the coarsening subtask. Given the value distribution at several time points, there is a need to calculate an aggregated distribution

Temporal Aggregation. Generating an aggregation of stochastic processes. The aggregation subtask abstracts along the subject axis. This subtask is aimed at the application of aggregation functions, such as minimum, maximum, average, etc. on stochastic processes.

Temporal Correlation. Calculating the correlation between two stochastic processes. The correlation subtask compares two temporal abstractions. This subtask is intended to mainly compare two populations, but should work the same when comparing different time periods of the same subject.

3 The PTA Property

The central property of the PTA method is based on the notion of *temporal field*, as defined below. Following this definition, the property states, that each unobserved state of some stochastic process is a linear combination of the temporal fields of the observed states of the process. Thus, the unobserved distribution of bone-marrow toxicity grades is a linear combination of all of the observed distributions, before and after it. A proper basis that will fit the requirements of the PTA property could be found in the following two known definitions.

Let $\{w_{ij}\}_{1\le i\le m,1\le j\le n}$ and $\{\mu_i\}_{1\le i\le m}$ be constants. The random variables X_i are said to have *multivariate normal distribution*, if:

$$X_i = \sum_j w_{ij} \cdot Z_j + \mu_i \quad , Z_j \sim Normal(0,1) \; . \tag{1}$$

A stochastic process $\{X_t: t \geq 0\}$ is called *Gaussian process* if each state X_t of the process has a multivariate normal distribution.

Calculating a depending variable given the independent variables as they appear in a multivariate distribution may imply a temporal *persistence* of the independent variables. However, allowing the observed states to induce a *field*[1] over its temporal environment could express temporal knowledge about the stochastic process in question, such as a *periodic behavior* or *change* of the observed states. Thus, for each stochastic process, a temporal field is induced by a time index, which formally means a function that maps time points to states of the stochastic process, as follows:

$$field_{\vec{X}}(t): T \rightarrow \mathbb{R}^{\Omega} \quad , X_t: \Omega \rightarrow \mathbb{R} \; . \tag{2}$$

For example, suppose a stochastic process with a periodic behavior and cycle length c. The temporal field of an observed state of such stochastic process could be as follows:

$$(field_{\vec{X}}(t_s))(t_i) = \sin\left(\frac{\pi}{c} \cdot (|t_i - t_s| \bmod c)\right) \cdot X_{t_s} \; . \tag{3}$$

A specific choice for the selection of the weights of the independent variables can be suggested. These weights should express the notion that the closer-in-time the observed state is – the more relevant it is. That is, the absolute time difference between a dependent state and its observed state should be inversely proportional to the weight of the latter when estimating the former. Therefore, there is a need to choose a monotonic decreasing function of absolute time differences between a dependent state and its inducing observed states. The weighting function is of the following form:

$$w_{\vec{X}}: \Delta T \rightarrow \mathbb{R} \; . \tag{4}$$

A natural nonlinear choice for the monotonic decreasing weighting function would be a normal density, where its variance (σ^2) determines the temporal tolerance of observed states of the stochastic process. Thus, w may hold:

$$w_{\vec{X}}(\Delta t) = f_W(\Delta t) \quad , W \sim Normal(0, \sigma^2) \; . \tag{5}$$

Uncertain Observations. Observed states of stochastic processes are distributed as a function of the test taken and the data itself. Typically, where states of stochastic processes have a normal distribution, the mean (expectation) of the state is the value sampled, and the variance is determined by the reliability or the precision of the test taken.

Prior Knowledge. Each stochastic process may have a prior knowledge of its typical state. Prior distribution is expressed by giving it the $-\infty$ time index for the temporal field inducer argument.

[1] In the sense of an electromagnetic field.

4 Mechanisms of the PTA Method

The main computational concept in our methodology is the PTA chain. A *PTA chain* is defined as the application of any subset of the following composition of subtasks, while preserving the relative order among them:

$$Coarsen \circ Correlate \circ Aggregate \circ Transform \circ Interpolate\,(data)\,. \quad (6)$$

Temporal Interpolation. The subtask of interpolation is solved by the application of the PTA property. Given the temporal weighting function of a stochastic process, its values need to be normalized to ensure they sum to unity. The subset of sampled states which participate in the calculation process of each unobserved state determines the precision of its distribution, and could be determined given the temporal weighting function. If we interpolate in t_i and have all of the points that are known t_s sampled, then:

$$X_{t_i} = \frac{1}{\sum_{t_s} w_{\vec{X}}(t_i - t_s)} \sum_{t_s} w_{\vec{X}}(t_i - t_s) \cdot (field_{\vec{X}}(t_s))(t_i)\,. \quad (7)$$

The procedure *Interpolate* fills the missing states of the given PTA chain. First, it retrieves all relevant raw data and extracts the empty states. Then, the interpolation function is applied to the extracted empty states. For the case in which updates to the underlying data occur, we consider a hierarchical system of states, where each unobserved state has a set of observed parent states, as depicted by Pearl [9]. In case the sample is updated, propagating the new piece of evidence we are viewing as the perturbation that propagated through a Bayesian network via message-passing between neighboring processors.

The knowledge required for the application of the interpolation subtask includes for each type of PTA chain the definitions of temporal fields (the default is set to persistence of the inducer state), temporal weighting (the default is set to normal density function with mean 0), prior distribution of a typical state (no default is set), and a function that maps each pair of test taken and datum (sampled value) to the distribution of the field inducing state (default sets sampled value to the state's mean).

Temporal Coarsening. The procedure *Coarsen* averages a PTA subchain. The value of such application is a coarser time-granularity state, according to the following formula:

$$X_{[t_i, t_j]} = \frac{1}{j - i + 1} \cdot \sum_{k=i}^{j} X_{t_k}\,. \quad (8)$$

Temporal Transformation. The procedure *Transform* returns the application of the given transformation function to the given PTA chains according to the following formula:

$$Y_t = (g(\vec{X}_1, \ldots, \vec{X}_n))(t)\,. \quad (9)$$

If g has the following form, then $|g|$ is called a *rate* transformation, and *sgn(g)* (positive, negative or zero) is called a *gradient* transformation:

$$(g(\overrightarrow{X}))(t_i)=\frac{X_{t_i}-X_{t_{i-1}}}{t_i-t_{i-1}}\ . \tag{10}$$

For example, a context of a Bone-Marrow Transplantation (BMT) is defined as the application of the following transformation function to the Boolean day-granularity stochastic process that represents a BMT:

$$(g(\overrightarrow{BMT}))(t)=BMT_{t-3}\vee\ldots\vee BMT_{t+90}\ . \tag{11}$$

In order to save extra computational costs, a time restriction of time-series of the arguments of each transformation is needed. This is accomplished by the definition and application of functions of the following form:

$$h_{\vec{Y}}(t)=\langle\overrightarrow{T}_1,\ldots,\overrightarrow{T}_n\rangle\ . \tag{12}$$

For example, in the case of a contemporaneous transformation of several arguments (e.g., height and weight) into a higher-level abstraction (e.g., body-mass index), the time-series of the arguments are the same as those of the abstraction.

Temporal Correlation. Applied to stochastic processes of different sample spaces, independent subjects or same, resulting in a series of correlation factors. This measure is computed as a time series of correlations between corresponding states of the given PTA chains:

$$\rho(X_{t_i},Y_{t_j})=\frac{Cov(X_{t_i},Y_{t_j})}{\sqrt{Var(X_{t_i})\cdot Var(Y_{t_j})}}\ . \tag{13}$$

An example for a single subject would be the contemporaneous correlations between height and weight or correlation of height during different periods for the same person.

Temporal Aggregation. Applied to stochastic processes of the same sample space and independent subjects, resulting in a new stochastic process. This measure is computed as a new PTA chain, where each of its state is the application of some aggregative function (minimum, maximum, average, etc.) to the corresponding states of the given PTA chains.

$$agg_{t_i}(\overrightarrow{X}_1,\ldots,\overrightarrow{X}_n)=agg(X_{1t_i},\ldots,X_{nt_i})\ . \tag{14}$$

5 Parallel Implementation

The computational model used to compute a PTA chain is *goal-driven, bottom-up* and *knowledge-based*. That is, the main algorithm is required to compute the result of a PTA chain (the goal), given the transformation and interpolation functions (the tem-

poral knowledge) as well as the access to the data, beginning at the raw (lowest abstraction level) data. The computational model is parallelized (and hence *scalable* [10]) in three orthogonal aspects: (1) Time, during the calculation of the PTA chains' states; (2) Transformation, during the calculation of the transformation arguments; and (3) Subject, during the calculation of the PTA chains for multiple subjects.

The PTA architecture is in the process of being fully implemented using the C++ programming language, the Standard Template Library (STL), and the MPICH2 implementation of the Message-Passing Interface (MPI)[2], an international parallel programming standard. The implementation is thus object-oriented and platform-independent. The implementation is in the process of being fully integrated into the IDAN system [11], which satisfies the need to access knowledge and data sources.

6 Discussion

In this paper, we proposed a probabilistic method to solve the task of abstraction of time-oriented records. The new method has removed several limitations of former methods. First, the use of PTA chains enables the expression of uncertainty in the underlying data as well as their derived abstractions. Second, two mechanisms were described for temporal abstraction of the data of multiple subjects. Third, the interpolation mechanism was shown to be sound and complete. However, observed data are assumed to be independently distributed. This assumption could be easily removed, given the necessary domain-specific conditional distribution functions.

The Markovian property (i.e., the conditional distribution of any future state, given the present state and all past states, depends only on the present state) is not assumed by the PTA method. Specifically, the temporal interpolation mechanism considers past states when computing future states, thus allowing for a broader set of domain-specific models of prediction knowledge, as well as for cases where the current state may be attributed a higher variance than states observed in the past. Naturally, this interpolation models entails a computational cost. However, the temporal interpolation of the PTA model is performed at the lowest abstraction level only (as opposed to being repeatedly performed at every abstraction level as in the KBTA method [1]) in a parallel manner, thus gaining a potential *efficiency* [12] in computing high-level abstractions. Finally, the components of the PTA method are highly modular and do not assume, for example, a particular temporal representation.

Acknowledgments

This work has been supported in part by NIH award No. LM-06806 and Israeli Ministry of Defense award No. 89357628-01. We would very much like to thank Denis Klimov and Efrat German of the Medical Informatics Research Center in Ben-Gurion University.

[2] http://www.mpi-forum.org/

References

1. Y. Shahar: A Framework for Knowledge-Based Temporal Abstraction. Artificial Intelligence (1997) 90:79-133
2. M.J. O'Connor, W.E. Grosso, S.W. Tu, M.A. Musen: RASTA: A Distributed Temporal Abstraction System to facilitate Knowledge-Driven Monitoring of Clinical Databases. MedInfo, London (2001)
3. A. Spokoiny, Y. Shahar: A Knowledge-based Time-oriented Active Database Approach for Intelligent Abstraction, Querying and Continuous Monitoring of Clinical Data. MedInfo (2004) 84-88
4. M. Balaban, D. Boaz, Y. Shahar: Applying Temporal Abstraction in Medical Information Systems. Annals of Mathematics, Computing & Teleinformatics (2004) 1(1):54-62
5. M.G. Kahn: Combining physiologic models and symbolic methods to interpret time varying patient data. Methods of Information in Medicine (1991) 30(3):167-178
6. I.J. Haimowitz, I.S. Kohane: Automated trend detection with alternate temporal hypotheses. Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence. Morgan Kaufmann, San Mateo (1993) 146-151
7. A. Salatian, J. Hunter: Deriving trends in historical and real-time continuously sampled medical data. Journal of intelligent information systems (1999) 13:47-71
8. Y. Shahar: Knowledge-Based Temporal Interpolation. Journal of Experimental and Theoretical Artificial Intelligence (1999) 11:123-144
9. J. Pearl: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann Publishers (1987)
10. K. Hwang, Z. Xu: Scalable Parallel Computing, WCB McGraw-Hill (1998)
11. D. Boaz, Y. Shahar: A Framework for Distributed Mediation of Temporal-Abstraction Queries to Clinical Databases: Artificial Intelligence in Medicine (in press)
12. R.P. Brent: The Parallel Evaluation of General Arithmetic Expressions. Journal of the ACM (1974)

Learning Classifiers Using Hierarchically Structured Class Taxonomies

Feihong Wu, Jun Zhang, and Vasant Honavar

Artificial Intelligence Research Laboratory,
Department of Computer Science, Iowa State University,
Ames, Iowa 50011-1040, USA
{wuflyh, jzhang, honavar}@cs.iastate.edu

Abstract. We consider classification problems in which the class labels are organized into an abstraction hierarchy in the form of a class taxonomy. We define a structured label classification problem. We explore two approaches for learning classifiers in such a setting. We also develop a class of performance measures for evaluating the resulting classifiers. We present preliminary results that demonstrate the promise of the proposed approaches.

1 Introduction

Machine learning algorithms to design of pattern classifiers have been well studied in the literature. Most such algorithms operate under the assumption that the the class labels are mutually exclusive. However, many real world problems present more complex classification scenarios. For instance, in computer vision application, natural scene containing multiple objects can be assigned to multiple categories [3]; in a digital library application, a text document can be assigned to multiple topics organized into a topic hierarchy; in bioinformatics, an ORF may have several functions [5]. In each of these cases, the class labels are naturally organized in the form of a hierarchically structured class taxonomy which defines an abstraction over class labels. Such a classification scenario presents two main challenges: (1) The large number of class label combinations make it hard to reliably learn accurate classifiers from relatively sparse data sets. (2) Standard metrics for evaluating classifiers in settings where class labels are mutually exclusive are not suitable for evaluation of classifiers in settings where the class labels are organized into a class hierarchy. Despite recent attempts to address some of these problems, [1, 2, 3, 4, 5, 6, 7], at present, a general solution is still lacking. Against this background, we explore approaches to learning classifiers in the presence of class taxonomies. The paper is organized as follows. Section 2 presents a precise formulation of the single label, multi label and the structured label classification problems; Section 3 describes two approaches to learning classifiers from data in the presence of class taxonomies; Section 4 explores performance measures for evaluating the resulting classifiers; Section 5, briefly describes results of experiments using the Reuters-21578 [8] data and genotype data [5]; Section 6 concludes with a summary and discussion.

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 313–320, 2005.

2 Preliminaries

Many standard classifier learning algorithms normally make the basic assumption of single label instances. That is, each instance that is represented by an ordered set of attributes $\mathbf{A} = \{A_1, A_2, ..., A_N\}$ can belong to one and only one class from a set of classes $\mathbf{C} = \{c_1, c_2, ..., c_M\}$. Therefore, class labels in $\mathbf{C}$ are mutually exclusive.

In multi label classification settings, class labels are not mutually exclusive. Each instance can be labelled using a subset of labels $c_s \subset \mathbf{C}$, where $\mathbf{C} = \{c_1, c_2, ..., c_M\}$ is a finite set of possible classes. If instances can be labelled with arbitrary subsets of $\mathbf{C}$, the total number of possible multi label combinations is 2^M.

An even more complex classification scenario is one in which instances to be classified are assigned labels from a hierarchically structured class taxonomy. Here, we define class taxonomy first and then formalize the resulting structured label classification problem.

Definition 1 (Class Taxonomy). *A Class Taxonomy $\mathcal{CT}$ is a tree structured regular concept hierarchy defined over a partially order set $(\boldsymbol{C}_T, \prec)$, where $\boldsymbol{C}_T$ is a finite set that enumerates all class concepts in the application domain, and relation $\prec$ represents the is-a relationship that is both anti-reflective and transitive:*

- *The only one greatest element "ANY" is the root of the tree.*
- *$\forall c_i \in \boldsymbol{C}$, $c_i \prec c_i$ is false.*
- *$\forall c_i, c_j, c_k \in \boldsymbol{C}$, $c_i \prec c_j$ and $c_j \prec c_k$ imply $c_i \prec c_k$.*

A tree structured class taxonomy represents class memberships at different levels of abstraction. The root of a class taxonomy is the most general label (i.e., "ANY") that is applicable to any instance. The leaves of class taxonomy indicate the most specific labels. The tree structure imposes strict constraints on these class memberships. Therefore, when an instance is assigned a label l from a hierarchically structured class taxonomy, it is implicitly labelled with all the ancestors of the label l in the class taxonomy.

Definition 2 (Structured label). *Any structured label $\mathcal{C}_s$ is represented by a subtree of $\mathcal{CT}$. $\mathcal{C}_s$ is a partially order set $(\boldsymbol{C}_s, \prec)$ that defines the same* is-a *relationships as in $\mathcal{CT}$. $\forall c_i \in \boldsymbol{C}_s$, c_i is ANY or $c_i \prec parent(c_i)$, where $parent(c_i) \in \boldsymbol{C}_s$ is the immediate ancestor of c_i in $\mathcal{CT}$.*

A class taxonomy imposes constraints on the integrity and validity of the structured labels. The integrity constraint states that $\mathcal{C}_s$ is a subtree structure of $\mathcal{CT}$ sharing the same root: Structured label is not an arbitrary fragment of the class taxonomy. The validity constraint captures the *is-a* relationships among class labels within a class taxonomy. A structured label is invalid if it contains a label l but not the parents of l in a given class taxonomy.

3 Methods

3.1 Binarized Structured Label Learning

One simple approach is to build a classifier consisting of a set of binary classifiers (one for each class). However, the drawbacks of this approach are obvious: (1) When making predictions for unlabelled instances, the classification results may violate the integrity and validity constraints. (2) The set of binary classifiers fails to exploit potentially useful constraints provided by the class taxonomy during learning.

To overcome these disadvantages, we build a hierarchically organized collection of classifiers that mirrors the structure of the class taxonomy $\mathcal{CT}$. The resulting classifiers form a partially ordered set $(h_{\mathcal{CT}}, \prec)$, where $h_{\mathcal{CT}} = \{h_{C_1}, \cdots, h_{C_M}\}$ is the set of classifiers, and $\prec$ represents partial orders among classifiers. If C_j is a child of C_i in $\mathcal{CT}$, then the respective classifiers satisfy the partial order $h_{C_j} \prec h_{C_i}$. This partial order on classifiers guides the classification of an instance. If $h_{C_j} \prec h_{C_i}$, an instance will not be classified using h_{C_j} if it has been classified as not belonging to C_i (i.e., output of h_{C_j} is 0). We call our method of building such hierarchically structured classifiers "Binarized Structured Label Learning" (BSLL).

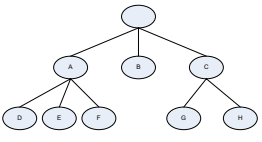


Fig. 1. Structure class taxonomy

3.2 Split-Based Structured Label Learning

A second approach to structured label learning is an adaptation of an approach to multi-label learning. We digress briefly to outline approaches to multi-label learning.

In real world applications it is very rare that each of the 2^M multi label combinations appear in the training data. The actual number of multi labels is much smaller than the possible number 2^M. Thus, we may set an upper limit on the number of possible class label combinations. If the number of labels that can occur in a multi-label is limited to 2, we will only consider the combinations of 2 class labels instead of M class labels. Another option is to consider only the multi labels that appear in the training data. In either case, we can not apply standard learning algorithms directly to the multi-label classification problem. This is because the multi label and the individual class labels are not mutually exclusive and it is not uncommon for some instances to be labelled with a single class label and others with multi labels. Because most standard learning algorithms assume mutually exclusive class labels, we will need to generate mutually exclusive classes. For example, consider $\mathbf{C} = \{A, B, C\}$

with instances set S_A, S_B, S_C respectively. Suppose the only multi label observed in the training data is $\{A, B\}$. Note that $S_A \bigcap S_B \neq \emptyset$. So the extended class label set is $\mathbf{C}' = \{\hat{A}, \hat{B}, \hat{C}, A\&B\}$,which represents instance set $S_A - S_A \bigcap S_B, S_B - S_A \bigcap S_B, S_C, S_A \bigcap S_B$.

This approach to transforming class labels to obtain mutually exclusive class labels can be applied to structured label learning problem by building split-based classifiers. We will first define a split in a class taxonomy $\mathcal{CT}$, and then for each split we show how to learn a respective classifier by learning from instances with extended label sets (as outlined above).

Definition 3 (Split). *A split is a one level subtree within a class taxonomy, which includes one parent node and all its children nodes, and the links between the parent node and children nodes.*

Obviously, the number of splits in the class taxonomy is smaller than the number of nodes. We can build a set of classifiers on the splits to solve structured label problem so to decrease the number of resulting classifiers. Within each split, the structured label problem will be reduced to a multi label problem, and we only need to consider the combinatorial extensions on class labels at that particular level. Additionally, the split-based classifiers are also partially ordered according to a given class taxonomy. Any instance to be classified will follow this topological order of the split-based classifiers: start from the classifier for the split at first position, continue to run a split-based classifier only when predicted to be "1" by the parent split-based classifier.

4 Performance Measure for Structured Label Classification

In single label classification, a loss function(like standard 0-1 loss function) $\mathbf{loss}(c_p, c_o)$ can be defined to evaluate the cost of misclassifying the instance with observed class label c_o to the predictive class label c_p. However, this approach is inadequate in a structured label problem in which there is a need to take into account the relationships between labels assigned to an instance. Here each label set corresponds to a subtree of the class taxonomy in structured label problem. We define a misclassification cost associated with the label set produced by the classifier relative to the correct label set (the correct structured label).

Definition 4 (Node Distance). *Node distance is a value $d(c_i, c_j)$ denoting the difference of labels c_i,c_j. It has the following properties:*

- $d(c_i, c_j) \geq 0$
- $d(c_i, c_j) = d(c_j, c_i)$
- $d(c_i, c_i) = 0$

Definition 5 (Dummy Label). *Dummy label θ is an"add-on" label to the class taxonomy which acts as a predicted value to the instance when a classifier can not decide the class label and does nothing. Thus this is a "label by default". It has the following properties:*

- $d(\theta, c_i) = d(\theta, c_j)$
- $d(c_i, c_j) \leq d(\theta, c_i)$

Definition 6 (Non-Redundant Operation). *A non-redundant operation (with Φ as the operator) to a label set C_i is to keep the children labels when both children labels and their parent labels are present, such that we eliminate the label redundancies within a class taxonomy.*

Definition 7 (Mapping). *A mapping f between two label sets $\mathbf{C}_1, \mathbf{C}_2$ with the same cardinality is a bijection $f : \mathbf{C}_1 \rightarrow \mathbf{C}_2$.*

We calculate the distance $d(\mathbf{C}_p, \mathbf{C}_o)$ between $\mathbf{C}_p$ and $\mathbf{C}_o$, the predicted and actual label (respectively) for each classified instance as follows:

- If the cardinalities of $\mathbf{C}_p$ and $\mathbf{C}_o$ are equal, find a mapping to minimize the sum of node distances and divide by the cardinality of the label sets to obtain the distance.
- If the cardinalities of the two label sets are not equal, add as many dummy labels θ as needed to the label set with fewer elements to make the cardinalities of the two label sets equal and then calculate the distance between the two label sets as before.

The performance of the classifier on a test set is obtained by averaging the distances between predicted and actual labels of instances in the test set T as follows: $\bar{d} = \frac{\sum_{\mathbf{T}} d(\mathbf{C}_p, \mathbf{C}_o)}{|\mathbf{T}|}$. The lower the value of this measure, the better the classifier (in terms of misclassification cost).

5 Experimental Results

Given a structured label data set, we need the pair-wise node distances between class labels to compute the misclassification cost as described above. These distances can be specified by a domain expert. Alternatively, the distances may be estimated from a training set based on cooccurence of class labels as follows: For each level in the class taxonomy, we calculate the occurrence of classes in the training set, divide it by the number of labels at that level of the class taxonomy. We calculate the distance between class labels as follows: We place the "add-on" label θ in the root node of the class taxonomy tree and set the edge distance as the level weight. For two nodes, if one is ancestor of the other, the node distance will be the sum of the edge distances along the path that connects them; if neither node is an ancestor of the other, the distance between them

is defined as the average distance of the two nodes from their nearest common ancestor. After normalization, we assign distance 1 to any two labels in the top level together with the "add-on" label θ, and the maximal node distance equals to the summation of all the level weights as 1.268 in Reuters-21578 data and 1.254 in phenotype data set.

5.1 Results on Reuters-21578 Data Set

Reuters-21578 data, originally collected by Carnegie Group for text categorization, does not have a predefined hierarchical class taxonomy. However, many documents are labelled with multiple topic classes. We extracted 670 documents. In this set, more than 72% of the documents have multiple class labels. We created a two-level class taxonomy using current categories of the documents as follows:

grain(barley,corn,wheat,oat,sorghum)
livestock(l-cattle,hog)

We used a Naive Bayes classifier as the base classifier and estimated the performance of the resulting structured label classifier using 5 fold cross validation. The results in tables 1, 2 suggest that binarized structured label learning performs as well as split-based structured label learning in this case. Both have good predictive accuracy for the classes that appear in the first level of the class taxonomy: grain, livestock. The overall performance of the two methods (as measured by the estimated misclassification cost) is slightly different, while the average recall and precision calculated over the entire class hierarchy are very close.

Table 1. Average distance: learning on Reuters-21578 data set

	binarization learning	split-based learning
d	0.217	0.251

Table 2. Recall&precision: learning on Reuters-21578 data set

	binarization learning		split-based learning	
	recall	precision	recall	precision
grain	0.993	0.964	0.993	0.968
livestock	0.766	0.893	0.752	0.917
barley	0.498	0.440	0.454	0.442
wheat	0.852	0.735	0.859	0.724
corn	0.839	0.721	0.818	0.726
oat	0.270	0.75	0.167	0.75
sorghum	0.408	0.560	0.324	0.591
l-cattle	0.146	0.417	0.167	0.339
hog	0.729	0.786	0.717	0.686

5.2 Results on Phenotype Data Set

Our second experiment used the phenotype data set introduced by Clare and King[5] whose class taxonomy is a hierarchical tree with 4 levels and 198 labels.

We choose the C4.5 decision tree as the base classifier to run the binarization learning and split-based learning in 5-fold cross validation. Split-based structured label learning shows better performance than binarized structured label learning on this data set. The misclassification cost is 0.79. The split-based structured label learning predicts 1 out of 4 class labels correctly in the 1_{st} level branches. Compared to the Reuters-21578 data set, the phenotype data set is much more sparse which might explain the fact that the results are not as good as in the case of the Reuters-21578 data set.

We also calculate accuracy, recall and precision of each class label. It turns out that the accuracy of each class label is quite high(95%). This is due to the fact that this data set is highly unbalanced and each classifier has a high true negative rate.Owing to the sparseness of the data set, many class labels do not appear in the test data set. This leads to undefined recall and precision estimates because of division by 0. Hence, only those class labels with recall and precision estimates available are listed in Figure 2. They show that split-based structured label learning performs better in terms of recall and precision, which is consistent with the relative performance of the two methods in terms of misclassification cost.

Table 3. Average distance: learning on phenotype data set

	binarization learning	split-based learning
d	1.171	0.790

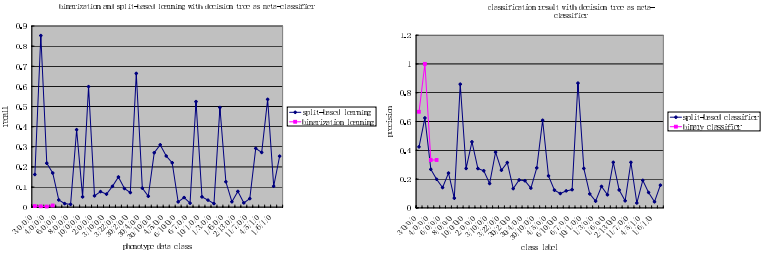


Fig. 2. Recall&precision: learning on phenotype data set

6 Summary and Discussion

In this paper, we have:

– Precisely formulated of learning from data using abstractions over class labels – the structured label learning problem – as a generalization of single label and multi label problems.

- Described two learning methods, binarized and split-based approaches to learning structured labels both of which can be adapted to work with any existing learning algorithm for single label learning task (e.g., Naive Bayes, Decision tree, Support vector machine, etc.).
- Explored a performance measure for evaluation of the resulting structured label classifiers.

Some directions for future work include:

- Development of algorithms to incorporate techniques for exploiting CT (class taxonomies) to handle partially specified class labels.
- Development of more sophisticated metrics for evaluation of structured label classifiers.

Acknowledgements. This research was supported in part by a grant from the National Institutes of Health (GM066387) to Vasant Honavar

References

1. A. McCallum "Multi label text classification with a mixture model trained by EM". AAAI'99 Workshop on Text Learning., 1999.
2. T,Joachims. "Text categorization with Support Vector Machines: Learning with many relevant features". In Machine Learning: ECML-98, Tenth European Conference on Machine Learning, pp. 137–142. 1998
3. X. Shen, M. Boutell, J. Luo, and C. Brown "Multi label Machine learning and its application to semantic scene classification" , in Proceedings of the 2004 International Symposium on Electronic Imaging (EI 2004), Jan. 18-22, 2004
4. H.-P. Kriegel, P. Kroeger, A. Pryakhin, and M. Schubert "Using Support Vector Machines for Classifying Large Sets of Multi-Represented Objects", in Proc. 4th SIAM Int. Conf. on Data Mining, pp. 102-114, 2004
5. A. Clare and R. D. King "Knowledge Discovery in Multi label Phenotype Data", 5th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD2001), volume 2168 of Lecture Notes in Arti cial Intelligence, pages 42-53, 2001
6. H. Blockeel, M. Bruynooghe, S. Dzeroski, J. Ramon, and J. Struyf. "Hierarchical Multi-Classification", Proceedings of the First SIGKDD Workshop on Multi-Relational Data Mining (MRDM-2002), pages 21–35, July 2002
7. K. Wang, S. Zhou, S.C. Liew, "Building hierarchical classifiers using class proximity", Technical Report, National University of Singapore, 1999
8. The Reuters-21578, Distribution 1.0 test collection is available from `http://www.daviddlewis.com/resources/testcollections/reuters21578`

Feature-Discovering Approximate Value Iteration Methods

Jia-Hong Wu and Robert Givan*

Electrical and Computer Engineering, Purdue University,
W. Lafayette, IN 47907, USA
{jw, givan}@purdue.edu

Abstract. Sets of features in Markov decision processes can play a critical role in approximately representing value and in abstracting the state space. Selection of features is crucial to the success of a system and is most often conducted by a human. We study the problem of automatically selecting problem features, and propose and evaluate a simple approach reducing the problem of selecting a new feature to standard classification learning. We learn a classifier that predicts the sign of the Bellman error over a training set of states. By iteratively adding new classifiers as features with this method, training between iterations with approximate value iteration, we find a Tetris feature set that outperforms randomly constructed features significantly, and obtains a score of about three-tenths of the highest score obtained by using a carefully hand-constructed feature set. We also show that features learned with this method outperform those learned with the previous method of Patrascu et al. [4] on the same SysAdmin domain used for evaluation there.

1 Introduction

Decision-theoretic planning and reinforcement-learning methods facing astronomically large state spaces typically rely on approximately represented value functions (see, e.g., [2, 7]). Many such approximate representations rely on an appropriate set of problem features; for example, by taking a weighted combination of the feature values as the value function [1]. Human engineering of the problem features used has repeatedly proven critical to the success of the resulting system. For example, in [7, 8], TD-gammon exploits human-constructed problem-specific features to achieve a playing strength that can compete with world-class players.

A Markov decision process (MDP) is a formal model of a single agent facing a sequence of action choices from a pre-defined action space, and living within a pre-defined state space. After each action choice is made, a state transition within the state space occurs according to a pre-defined, stochastic action-transition model. The agent receives reward after each action choice according to the state visited (and possibly the action chosen), and has the objective of accumulating as much reward as possible (possibly favoring reward received sooner, using discounting, or averaging over time, or requiring that the reward be received by a finite horizon).

* Many thanks to Alan Fern for very useful discussions and input.

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 321–331, 2005.

MDP solution techniques often critically rely on finding a good *value function*; this is a mapping from states to real numbers with the intent that desirable states receive high values. Informally, a good value function should respect the action transitions in that good states will either have large immediate rewards or have actions available that lead to other good states; this property is formalized in *Bellman equations* that define the optimal value function (see below). The degree to which a given value function fails to respect action transitions in this way, formalized below, is referred to as the *Bellman error* of that value function.

Unfortunately, virtually every interesting MDP problem has an extremely large state space, preventing direct table-based representation of the value function. As a result, abstraction, approximation, and problem reformulation play a critical role in successfully representing and finding good value functions. A typical approach is to find useful *features*, which also map the state space to real numbers, and take the value function at each state to be a weighted combination of the features at that state. Here, good values for the weights can often be found using machine learning techniques involving search and gradient descent. In this paper, we address the problem of finding good features automatically: in most previous work the features are simply selected by the human designer. Learning features for this purpose can be regarded as learning an abstraction of the MDP state space: differences between states with the same feature values have been abstracted away.

Here, we study the problem of automatically selecting problem features for use in approximately representing value in Markov decision processes. We focus our initial work on this problem on binary features, i.e., mappings from state to Boolean values. We view each such feature as a set of states, those states where the feature is true.

We propose and evaluate a simple, greedy approach to finding new binary features for a linear-combination value estimate. Our heuristic approach assumes an initial "base" value estimate described by a linear approximation where the weights have already been tuned to minimize Bellman error. We attempt to reduce the Bellman error magnitude of this value estimate further by learning a new feature that is true in statespace regions of positive statewise Bellman error and correspondingly false in regions of negative statewise Bellman error, or vice versa. The learning problem generated is a standard supervised classification problem, and for this work we address this problem using the decision-tree learner C4.5 [5].

One view of this approach is that we are conducting approximate value iteration with an added mechanism for extending the available feature set. Given an initial feature set, imagine a sufficient period of approximate value iteration (or any similar weight adjustment method) to achieve convergence of the approximation to a value function $\tilde{V}$. We can think of the approximate value iteration process as "stuck", in that it can represent $\tilde{V}$ but not the Bellman update of $\tilde{V}$. (Of course, this assumes that the inductive updates being performed would find $\tilde{V}$ if they could represent it, which is only heuristically true.) We are then trying to induce features to enable representation of the Bellman update of $\tilde{V}$, so that the approximate value iteration process can continue to reduce Bellman error, with the larger feature space.

If the learner succeeds in capturing features that describe the statespace regions of positive and negative Bellman error, we can guarantee that adding these features makes

available weight assignments closer to the Bellman update of the base value estimate. Our practical method retrains the weights including the new feature(s), using approximate value iteration (AVI), and then repeats the process of selecting a new feature.

We have found surprisingly little previous work on selecting features automatically in MDPs. Patrascu et al. [4] give a linear programming technique for selecting new features. In their work, the primary technique for selecting weights is approximate linear programming (ALP). It is observed in [4] that ALP "only minimizes L_1 error"; perhaps for this reason, that work proposes to construct features aimed to minimize the L_1 error of the resulting approximation. It is stated there that there is a "hope that this leads to a reduction in Bellman error as a side effect."

Our technique works instead directly to reduce the Bellman error magnitude of the resulting approximation by trying to identify regions that contribute to large statewise Bellman error magnitude. Because the Bellman error at any given state is easily computable in many domains of interest, unlike the L_1 error, we are able to convert the problem of minimizing Bellman error magnitude to a supervised classification problem.

Patrascu et al. [4] do not mention any reduction to supervised-classification learning, nor is it clear how to construct such a reduction from the approach they describe, because the computation of the L_1 error for particular states requires knowledge of the optimal value of those states. Thus, although decision trees are suggested as a possible representation for candidate features in that work, there is no suggestion that these trees be acquired by a supervised classification method like C4.5 and no discussion of how to overcome the need to know the optimal value if the L_1 error is to be used for supervision.

For another view of the contrast between our method and that previous method, consider that we seek a feature, via supervised classification, that corresponds to the region of states that have significantly positive (or alternatively, negative) Bellman error. There is no such declarative characterization of the desired feature in the Patrascu et al. work; rather, the region sought is that region that results in the best improvement in L_1 error after retraining via ALP (or less expensive-to-compute approximations of this "error after retraining"). This does not represent a reduction to classification and is a substantially different approach, using L_1 error after retraining (or a less expensive stand-in) as a scoring function.

Both approaches are reasonable, of course. We show below that our technique empirically outperforms this previous work on the planning domain used in their evaluation. Specifically, we require fewer new features to achieve the same Bellman error, and can achieve a lower overall Bellman error given enough features.

There is related previous work on function approximation in which new features are automatically added during supervised learning of real-valued functions [9]. It would be reasonable to consider, in comparison to our work here, plugging in such a function approximator at the learning step in approximate value iteration—this would result in an overall method similar in spirit to what we design directly here. We have not done an empirical comparison along these lines at this time.

We also evaluate our technique in the computer-game domain of Tetris. Starting from a constant value function based on only the uniformly true feature, our technique can add features automatically to produce performance that is significantly better than

a randomly constructed feature set, and is at about three-tenths of the performance of a carefully hand-constructed feature set.

In what follows, we first provide technical background on Markov decision processes and value-function approximation, then describe our technique for inducing new features to reduce approximation error, and finally present empirical results on two domains showing improvement over the state of the art, before concluding.

2 Technical Background

2.1 Markov Decision Processes

We define here our terminology for Markov decision processes. For a more thorough discussion of Markov decision processes, see [2] and [7]. A Markov decision process (MDP) D is a tuple (S, A, R, T) where state space S is a finite set of states, action space A is a finite set of actions, $R : S \times A \times S \rightarrow \mathbb{R}$ is the reward function, and $T : S \times A \rightarrow \mathcal{P}(S)$ is the transition probability function that maps (state, action) pairs to probability distributions over S. $R(s_1, a, s_2)$ represents how much immediate reward is obtained by taking action a from state s_1 and ending up in state s_2. $T(s_1, a, s_2)$ represents the probability of ending up in state s_2 if the action a is taken from state s_1.

A *policy* π for an MDP is a mapping $\pi : S \rightarrow A$. Given policy π, the value function $V^\pi(s)$ gives the expected discounted reward obtained starting from state s and selecting action $\pi(s)$ at each state encountered. Rewards after the first time step are discounted by a factor γ where $0 \leq \gamma < 1$. A Bellman equation relates V^π at any state s and successor states s':

$$V^\pi(s) = \sum_{s' \in S} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^\pi(s')].$$

There is at least one optimal policy π^* for which $V^{\pi^*}(s)$, abbreviated $V^*(s)$, is no less than $V^\pi(s)$ at every state s, for any other policy π. Another Bellman equation governs V^*:

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V^*(s')].$$

From any value function V, we can compute a policy Greedy(V) that selects, at any state s, the greedy look-ahead action $\arg\max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V(s')]$. The policy Greedy$(V^*)$ is an optimal policy. *Value iteration* iterates the operation $V'(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V(s')]$, computing V' from V, producing a sequence of value functions converging to V^*, regardless of the initial V used.

We define the statewise Bellman error $B(V, s)$ for a value function V at a state s to be $V'(s) - V(s)$. We will be inducing new features based on the sign of the statewise Bellman error. The sup-norm distance of a value function V from the optimal value function V^* can be bounded using the Bellman error magnitude, which is defined as $\max_{s \in S} |B(V, s)|$ (e.g., see [11]).

Linear Approximation of Value Functions. We assume that the states of the MDP have structure. In particular, we assume a state is a vector of basic properties with Boolean,

integer, or real values, and that the state space is the set of all such vectors. We call these basic properties *state attributes*. This factored form for states is essential to enable compact representation of approximate value functions.

A common solution to the problem of representing value functions (e.g., value iteration) in very large, structured state spaces is to approximate the value $V(s)$ with a linear combination of features extracted from s, i.e., as $\tilde{V}(s) = \sum_{i=0}^{p} w_i f_i(s)$, where w_i is a real-valued *weight* for the ith feature $f_i(s)$. Our goal is to find features f_i (each mapping states to boolean values) and weights w_i so that $\tilde{V}$ closely approximates V^*.

Many methods have been proposed to select weights w_i for linear approximations [6, 10]. Here, we use a trajectory-based approximate value iteration (AVI) approach. Other training methods can be substituted and this choice is orthogonal to our main purpose.

The AVI method we deploy constructs a fixed-length sequence of value functions $V^1, V^2, \ldots, V^T$, and returns the last one. Each value function V^β is defined by weight values $w_0^\beta, w_1^\beta, \ldots, w_p^\beta$ as $V^\beta(s) = \sum_{i=0}^{p} w_i^\beta f_i(s)$. Value function $V^{\beta+1}$ is constructed from V^β by drawing a training set of trajectories[1] under the policy Greedy(V^β) and updating the weights according to this training set as follows.

Let $s_1, s_2, \ldots, s_n$ be a sequence of training states, which we generate by drawing a set of trajectories under the current greedy policy. A training set for weight adjustment is defined as $\{(s_j, V'(s_j)) \mid 1 \leq j \leq n\}$. We adjust weights in iterations of batch training. In the l'th batch training iteration, the weight update for the i'th weight in the training set is $w_{i,l+1} = w_{i,l} + \frac{1}{n} \sum_j \alpha f_i(s_j)(V'(s_j) - \sum_{i=0}^{p} w_{i,l} f_i(s_j))$. Here, α is the learning rate parameter. We take the initial weights $w_{i,1}$ to be w_i^β. After κ iterations we set $w_i^{\beta+1} = w_{i,\kappa+1}$.

2.2 Decision Tree Classification

A detailed discussion of classification using decision trees can be found in [3]. A decision tree is a binary tree with internal nodes labelled by state attributes (and, in our case, learned features), and leaves labelled with classes (in our case, either zero or one). A path through the tree from the root to a leaf with label l identifies a partial assignment to the state attributes—each state consistent with that partial assignment is viewed as labelled l by the tree. We learn decision trees from training sets of labelled states using the well known C4.5 algorithm [5]. This algorithm induces a tree greedily matching the training data from the root down. We use C4.5 to induce new features—the key to our algorithm is how we construct suitable training sets for C4.5 so that the induced features are useful in reducing Bellman error.

3 Feature Construction for MDPs

We propose a simple method for constructing new features given a current set of features and an MDP for which we desire an approximation of V^*. We first use AVI, as

[1] The source of this set is a parameter of the algorithm, and it could for example be drawn by sampling initial states from some state distribution and then simulating π to some horizon from each initial state.

described above, to select heuristically best weights to approximate V^* with $\tilde{V}$ based on the current feature set. We then use the sign of the statewise Bellman error at each state as an indication of whether the state is undervalued or overvalued by the current approximation. If we can identify a collection of undervalued states (ideally, all such states) as a new feature, then assigning an appropriate positive weight to that feature should reduce the Bellman error magnitude. The same effect should be achieved by identifying overvalued states with a new feature and assigning a negative weight. We note that the domains of interest are generally too large for statespace enumeration, so we will need classification learning to generalize the notions of overvalued and undervalued across the statespace from training sets of sample states. Also, to avoid blurring the concepts of overvalued and undervalued with each other, we discard states with statewise Bellman error near zero from either training set.

More formally, we draw a training set of states Σ from which we will select training subsets Σ_+ and Σ_- for learning new features. The training set Σ can either be drawn uniformly at random from the state space, or drawn by collecting all states in sample trajectories starting at uniformly random start states under a policy of interest (typically Greedy($\tilde{V}$)). If using trajectories, each trajectory must be terminated at some horizon. The horizon and the size of Σ are parameters of our algorithm.

For each state s in Σ, we compute the statewise Bellman error $B(\tilde{V}, s)$. We then discard from Σ those states s with statewise Bellman error near zero, i.e., those states for which $|B(\tilde{V}, s)| < \delta$ for a non-negative real-valued parameter δ, and then divide the remaining states into sets Σ_+ and Σ_- according to the sign of $B(\tilde{V}, s)$. So, Σ_+ is the set $\{s|B(\tilde{V}, s) \geq \delta\}$ and Σ_- is the set $\{s|B(\tilde{V}, s) \leq -\delta\}$.

We note that computing statewise Bellman error exactly can involve a summation over the entire state space, whereas our fundamental motivations require avoiding such summations. In many MDP problems of interest, the transition matrix T is sparse in a way that set of states reachable in one step with non-zero probability is small, for any current state. In such problems, statewise Bellman error can be computed effectively using an appropriate representation of T. More generally, when T is not sparse in this manner, the expectation can be effectively approximately evaluated by sampling next states according to the distribution represented by T.

We then use Σ_+ as the positive examples and Σ_- as the negative examples for a supervised classification algorithm; in our case, C4.5 is used. The hypothesis space for classification is built from the primitive attributes defining the state space; in our case, we use decision trees over these attributes. We can also interchange the roles of Σ_+ and Σ_-, using the latter as positive examples. In our experiments, we do this interchanging for every other feature constructed.

The concept resulting from supervised learning is then treated as a new feature for our linear approximation architecture, with an initial weight of zero. The process can then be repeated, of course, resulting in larger and larger feature sets, and, hopefully, smaller and smaller Bellman error magnitude.

To conclude our description of our algorithm, we discuss setting the parameter δ dynamically, once in each iteration of feature construction. Rather than directly specify δ, we specify δ in terms of the standard deviation σ of the statewise Bellman error over the same distribution used in selecting states for the training set Σ. The value of σ is

easily estimated by sampling the training distribution and computing the Bellman error. We then set δ, at each iteration, to be a fixed multiple η of σ. This approach removes δ as a parameter of the algorithm, replacing it with the parameter η. This dynamic selection of δ allows adaptation to the decrease in Bellman error magnitude over the run of the algorithm.

4 Experiments

In this section, we present some experimental results for our feature construction algorithm. We use two domains in the experiments. The first domain is an 8×8 game of Tetris (Tetris). The second domain is a computer network optimization problem called SysAdmin, which we use primarily in order to compare to the closest previous related work; that work [4] used SysAdmin as a testing domain. Both the state attributes and the learned features in the experiments are binary features.

Tetris. For the Tetris domain, we start with 71 state attributes; 64 attributes which represent if the 64 squares are occupied or not, and 7 attributes which represent which of the 7 pieces is currently being dropped. We select training sets for feature construction by drawing trajectories from an initial state with an empty board and collecting 600,000 states on these trajectories as Σ. The training sets for AVI are selected by drawing 100 trajectories from an initial state with an empty board and allowing each trajectory to extend to the end of the game. We draw the trajectories using the Greedy($\tilde{V}$) policy. The discount factor γ is 0.9 for this experiment, and the parameter η is set to 0.3. In addition, κ is fixed at 100 and α at 0.01. AVI is stopped (appearing to have converged) after 1,200 training sets are drawn; at that point, a new feature is learned.

The results are shown in Figure 1. The score is determined by the average number of lines erased during a sequence of games. The performance of the learned features are evaluated by the 2,000-game average score for Greedy($\tilde{V}$) using the weights learned by AVI. Figure 1 displays the average of such evaluations over 4 separate trials of feature learning. In addition, we also show in Figure 1 the result of using sets of randomly generated features; such features are generated following the same procedure described in the previous section, but label examples in the training set Σ randomly instead of deciding the labeling by statewise Bellman error. Value functions constructed from randomly generated features perform poorly, and do not show improvement as the number of features used increases. Thus, our use of statewise Bellman error to label the training examples plays an important role in the performance of our feature construction algorithm.

We also tested AVI on human-constructed features in this domain. The features we used in this case were provided by Bertsekas in [2]. These features are useful features as considered by a human, and according to [2] they were selected after some testing. We tested the performance of the weights learned after each AVI iteration by running 2,000 games and taking the average score. The maximum 2,000-game average performance was 92.9, which was achieved after 22,634 iterations of AVI. This performance was substantially better than the best performance our learned feature set exhibited, which was 27.6 (using 34 learned binary features).

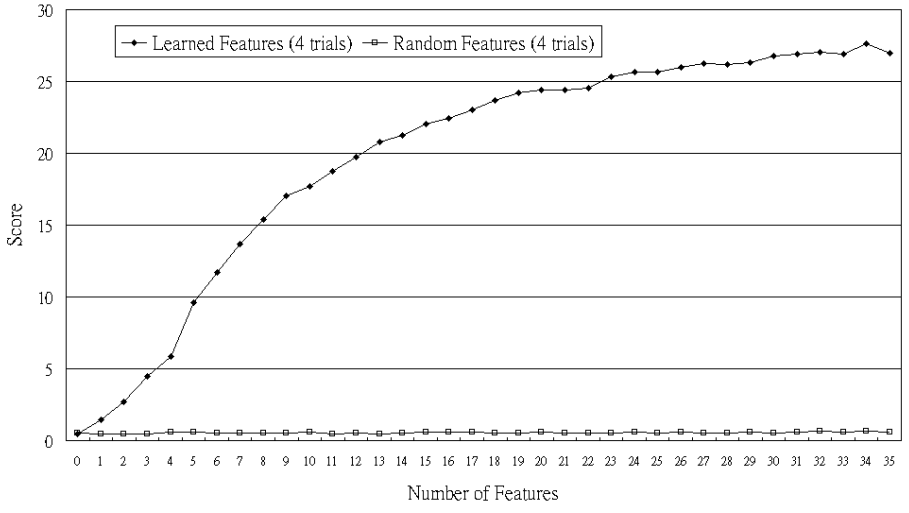


Fig. 1. Score (average number of lines erased in 2,000 games) plot for the learned features and randomly generated features in the 8×8 Tetris domain. For reference, the maximum score for the human-selected feature set from [2] was 92.9

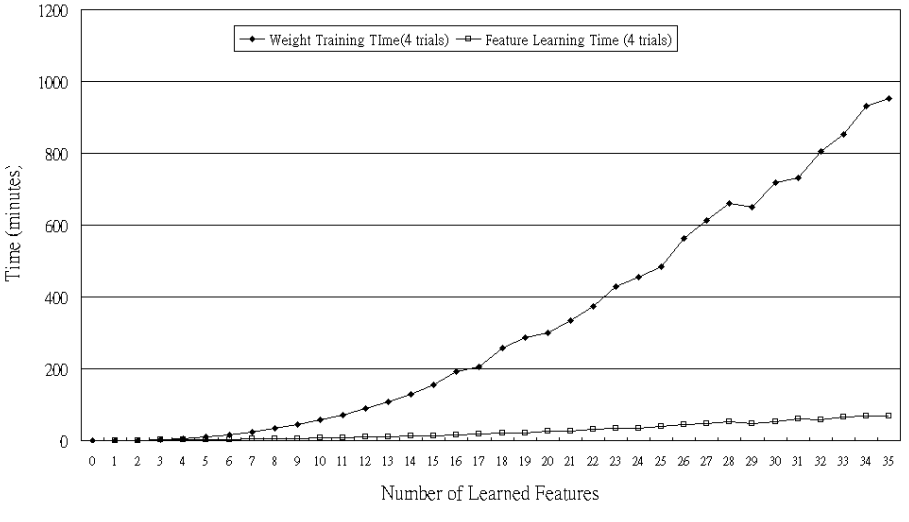


Fig. 2. Number of minutes required to generate the n'th feature, and to train the weights after the n'th feature is added to the feature set

We note that the human-selected features are all integer-valued, apparently giving the human set a clear advantage over our binary features (especially per feature). Clearly one approach for further improvement in feature learning is to design a feature-learning approach that can produce integer-valued features.

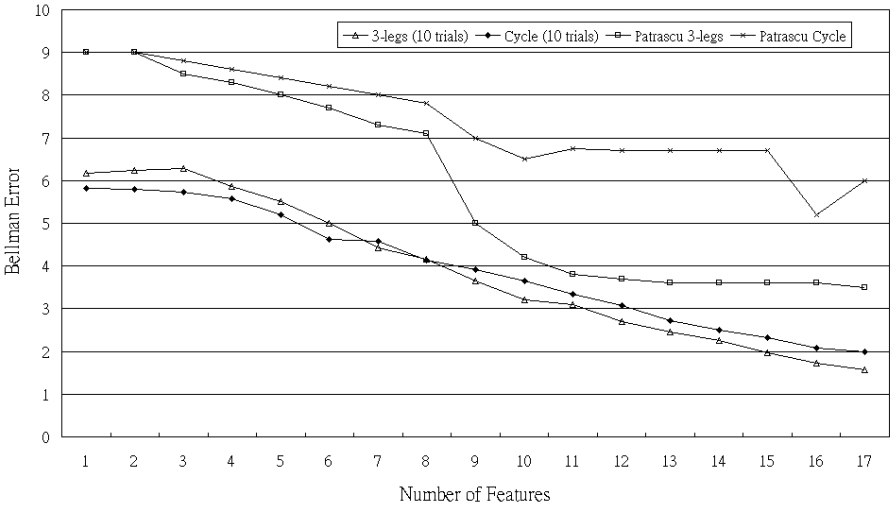


Fig. 3. Bellman error for SysAdmin domain (10 nodes)

We also study the the runtime behavior for our algorithm in this domain. We show in Figure 2 the execution time for generating the features and adjusting the weights. How long it takes a feature to be learned depends on how many training examples are collected and how many features exist. Since the number of training examples does not grow as we learn additional features, the time required to learn a new feature is eventually dwarfed by growing time required to train the weights for the growing feature set.

SysAdmin. For the SysAdmin domain, two different kinds of topologies are tested: 3-legs and cycle. There are 10 nodes in each topology. We follow the settings used in [4] for testing this domain. The target of learning in this domain is to keep as many machines operational as possible, since the number of operating machines directly affects the reward for each step. Since there are only 10 nodes, the on/off status of each node is used as a basic feature, which means there are a total of 1024 states. We simply use all states as the training set for feature construction. To enable direct comparison to the previous work in [4], we use Bellman error magnitude to measure the performance of the feature construction algorithm here.

For the experiments that use the whole state space as a training set, the plot of average Bellman error for 10 separate trials over the number of features learned is shown in Figure 3. We used γ equal to 0.95, η equal to 1, α equal to 0.1, and κ equal to 100. In this experiment there are 50 trajectories drawn in each AVI training set, each drawn from a random initial state, and using trajectory length 2. AVI was stopped, appearing to have converged, after 1,000 iterations.

Also included in Figure 3 are the results from [4]. We select the best result they show (from various algorithmic approaches) from the 3-legs and cycle domains shown in their paper (their "d-o-s" setting for the cycle domain and their "d-x-n setting" for the 3-legs domain).

Compared to the results in [4], also shown here, our feature construction algorithm achieves a lower Bellman error magnitude in these domains for the same number of features, throughout, and a lower converged Bellman error magnitude when new features stop improving that measure. This is another encouraging result for this proposed feature construction algorithm.

5 Conclusions and Future Work

From the experiments, the results show that our feature construction algorithm can generate features that show significantly better performance in 8×8 Tetris than randomly constructed features, and can produce features that outperform the features produced by the algorithms in [4] for the SysAdmin domain. However, our algorithm cannot learn a feature set for 8×8 Tetris that competes well with the human-constructed feature set provided in [2].

Our technique depends critically on the generalization ability of the classification learner to cope with large state spaces. The features generated by the feature-construction algorithm are currently represented as decision trees. Although the experiments showed that these features are useful in some problems, they are still hard to interpret. One goal for designing a good feature-construction algorithm is to be able to produce features that are understandable by humans. Furthermore, decision tree learning might not be adequate to find good generalizations in complex domains. We observed that the human-constructed features in [2] can be represented compactly using relational languages. Some of them, e.g. the number of "holes" in Tetris, are quite awkward to represent using the decision tree structure in this paper. One way we are considering for improving our algorithm is to use a relational classification or function-approximation algorithm combined with an expressive knowledge representation instead of using C4.5 with decision trees.

We note that the performance of machine-learned feature set appears to converge, with little added benefit per new feature, at a point where the policy corresponding to the learned value function is far short of optimal. This suggests a lack of state-space exploration during the feature learning stage. Another direction we are considering for improving our algorithm is to develop new exploration strategies for generating the training sets of states for feature learning.

References

1. R. Bellman, R. Kalaba, and B. Kotkin. Polynomial approximation – a new computational technique in dynamic programming. *Math. Comp.*, 17(8):155–161, 1963.
2. D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.
3. T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
4. R. Patrascu, P. Poupart, D. Schuurmans, C. Boutilier, and C. Guestrin. Greedy linear value-approximation for factored markov decision processes. In *AAAI*, 2002.
5. J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
6. R. S. Sutton. Learning to predict by the methods of temporal differences. *MLJ*, 3:9–44, 1988.

7. R. S. Sutton and A. G. Barto. *Reinforcement Learning*. MIT Press, 1998.
8. G. Tesauro. Temporal difference learning and td-gammon. *Comm. ACM*, 38(3):58–68, 1995.
9. P. E. Utgoff and D. Precup. Constuctive function approximation. In Motoda and Liu, editors, *Feature extraction, construction, and selection: A data-mining perspective*, pages 219–235. Kluwer, 1998.
10. B. Widrow and M. E. Hoff Jr. Adaptive switching circuits. *IRE WESCON Convention Record*, pages 96–104, 1960.
11. R. J. Williams and L. C. Baird. Tight performance bounds on greedy policies based on imperfect value functions. Technical report, Northeastern University, 1993.

Designing Views to Efficiently Answer *Real* SQL Queries

Foto Afrati[1], Rada Chirkova[2], Manolis Gergatsoulis[3], and Vassia Pavlaki[1]

[1] Department of Electrical and Computing Engineering,
National Technical University of Athens (NTUA),
15773 Athens, Greece*
{afrati, vpavlaki}@softlab.ntua.gr
[2] Computer Science Department, North Carolina State University,
Campus Box 7535, Raleigh, NC 27695-7535 **
chirkova@csc.ncsu.edu
[3] Department of Archive and Library Sciences, Ionian University,
Palea Anaktora, Plateia Eleftherias, 49100 Corfu, Greece
manolis@ionio.gr

Abstract. The problem of optimizing queries in the presence of materialized views and the related view-design problem have recently attracted a lot of attention. Significant research results have been reported, and materialized views are increasingly used in query evaluation in commercial data-management systems. At the same time, most results in the literature assume set-theoretic semantics, whereas SQL queries have bag-theoretic semantics (duplicates are not eliminated unless explicitly requested). This paper presents results on selecting views to answer queries in relational databases under set, bag, and bag-set semantics. The results can be used under each of the three assumptions, to find sound and complete algorithms for designing views and rewriting queries efficiently.

1 Introduction

A lot of work has been done recently on optimizing queries in the presence of materialized views. In this context, problems such as definition of views, composition of views, maintenance of views have been researched. At the same time, the majority of the research assumes set-theoretic semantics, while SQL queries have bag-theoretic semantics, where duplicates are not eliminated unless explicitly requested. As SQL is the query language used in most commercial database-management systems (DBMS), results on rewriting queries under bag or bag-set semantics are useful in practice.

* The project is co-funded by the European Social Fund (75 %) and National Resources (25%)-EPEAEK, Herakleitos.

** This author's work on this material has been supported by the National Science Foundation under Grant No. 0307072.

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 332–346, 2005.

The problem of view selection has received significant attention in the literature [3, 4, 12, 13, 14, 15, 17, 20, 21]. In this paper we consider view selection under set, bag, and bag-set semantics. The problem is as follows: Given a set of queries (which we call a *query workload*), a database, and a set of constraints on materialized views (e.g., *storage limit*, which is a bound on the amount of disk space available for storing the materialized views), return definitions of views that, when materialized in the database, would satisfy the constraints and reduce the evaluation costs of the queries. The original motivation for the view-selection problem comes from data-warehouse design, where we need to decide which views to store in the warehouse to obtain optimal performance [4, 15, 20]. Another motivation is provided by recent versions of several commercial DBMS, which support incremental updates of materialized views and use materialized views to speed up query evaluation. Choosing an appropriate set of views to materialize in the database is crucial in order to obtain performance benefits from these new features [3]. The view-selection problem and its generalizations will play an even greater role in contexts where data needs to be placed intelligently over a wide-area network, such as in peer-based data management [11].

Database relations are often duplicate-free. More precisely, database relations are often sets, while views and queries are often bags, defined without using the DISTINCT keyword (bag-set semantics). The bag-set semantics case is arguably more practical than the bag-semantics case, as relational database-management systems typically compute query answers using operators with bag-valued outputs on set-valued databases. At the same time, studying the bag-semantics case is important not just from the theoretical but also from practical perspective, as in view selection it is possible to design and materialize bag-valued views and thus to obtain bag-valued databases of stored data. Computing query answers on such databases using the rules of evaluating SQL queries on relational databases obeys the laws of bag, rather than bag-set, semantics.

We now give examples that illustrate the high complexity of the problem of selecting views to materialize when set semantics are assumed. In this paper we show that under bag or bag-set semantics the complexity of the view-selection problem is significantly lower, and thus it is more likely to find efficient view-selection algorithms that output "more optimal" views than in the set-semantics case. In the following, we make a distinction between views that contribute to the construction of tuples in the query answers — we call them *containment-target views* — and optional *filtering views* [2, 19] that may improve the efficiency of query processing; see Example 2 for more details.

Our first example shows that the search space of potentially useful views can be very large even for simple and common select-project-join queries.

Example 1. We exhibit a workload of select-project-join queries and a storage limit, such that it is not possible to materialize the answers to all the workload queries. We consider for materialization select-project-join views, such that each view alone satisfies the storage limit and can support all workload queries. This

example shows that for the given workload and under *set* semantics, at least an exponential (in the size of the query workload) number of such views have more subgoals than any query in the input workload.[1]

Consider a large retail chain with multiple stores and warehouses, where products are ordered and shipped daily from the warehouses to replenish the inventory in the stores. Suppose that the database has a `Shipments` relation, and let `warehouseID, warehouseCity, storeID, storeCity`, `orderNumber`, `shipmentNumber`, and `shipmentDate` be some of its attributes. Suppose the employees of the retail chain contract shipments to independent truck drivers, by attracting them with tours connecting two or more cities. The company predefines a number of tour types to offer to the truck drivers, and the company employees need to query the database and find out whether the tour requested by the driver exists starting at a given city. Every tour type starts and ends in the *same* city. The simplest tour is the "two-city roundtrip": The query returns all cities `warehouseCity` such that there exist two scheduled deliveries: one from `warehouseCity` to some `storeCity` on a given `shipmentDate 'date1'`, and the other from (a warehouse in) `storeCity` back to (some store in) `warehouseCity` on a later `shipmentDate 'date2'`. We now give a SQL definition of the query; note the `DISTINCT` keyword that enforces set semantics.

```
SELECT DISTINCT S1.warehouseCity FROM Shipments S1, Shipments S2
WHERE S1.storeCity = S2.warehouseCity AND S1.warehouseCity = S2.storeCity
AND S1.shipmentDate = 'date1' AND S2.shipmentDate = 'date2';
```

As we show formally in the extended version of this paper, under set semantics, for a given storage limit and for a query workload that has several such tour queries for tours of different lengths, we can select and materialize a *single* view such that the number of copies of the `Shipments` relation in the `FROM` clause of the view is *exponential* in the combined size of the query workload. Thus, in view selection under set semantics we potentially need to consider up to an exponential number of views in the size of the input query workload. At the same time, as shown in this paper (see Theorems 4 and 7), in view selection under bag or bag-set semantics we do not need to consider views whose definitions have more subgoals than the number of subgoals of the longest input query.

Our second example shows that even if we further simplify the language of queries of interest, there still remains a large number of views that could significantly reduce the evaluation costs of the queries under set semantics.

Example 2. We use here the application domain of Example 1. Suppose that in addition to the `Shipments` relation described in Example 1, the database also has an `Inventory` relation, where the attributes of interest to us are `storeID, productID, orderNumber,` and `isOutstanding`. We assume that the volume of daily orders to replenish store inventories is large, and that a single order

[1] This example is a variation on Example 1 in [8]; unlike that example, here we restrict the search space of views in that we do not consider filtering views.

is typically made for a large range of product types and potentially for several stores in the same area. Further, suppose that different product types are stored at different warehouses, and thus all the products ordered in a single order can be delivered to a store via multiple shipments. Finally, we assume that on delivery, the contents of most — but not all — shipments are put on the store shelves and reflected in the inventory records on the same day.

Suppose that at the end of each day, the management of the retailer chain routinely runs certain "daily report" queries on the deliveries. One of the queries asks for the stores and product IDs such that some quantity of the product has been ordered and was to be delivered to the store on the day in question, but the order is still listed as "outstanding" in the store inventory. The query in SQL is:

```
SELECT DISTINCT I.storeID, productID FROM Shipments S, Inventory I
WHERE S.storeID = I.storeID and S.orderNumber = I.orderNumber
AND shipmentDate = 'today' AND isOutstanding = 'yes';
```

Note that unlike the queries in Example 1, the `FROM` clause of this query has just one copy of each relation. That is, the daily-report query *has no self-joins*. We show in this paper (see Sections 3.1, 4.1, and 5.1) that even under set semantics (as well as under bag and bag-set semantics), when doing view selection for such queries we do not need to consider views whose definitions have more subgoals than the input queries. At the same time, we still need to consider a significant number of views. Under the domain assumptions in this example, even using reasonable indexes on `Shipments` or `Inventory` does not eliminate most redundant tuples in the result of the join, and thus postprocessing of a large temporary join result (which by definition never even has an index) is part of any query plan that does not use materialized views. (The reason is, under our domain assumptions a shipment where `shipmentDate = 'today'` typically corresponds to a large number of product IDs in the `Inventory` relation, where the value of `isOutstanding` is `'no'` for most records. Conversely, for a large number of combinations of values of `storeID` and `productID` in the `Inventory` relation where `isOutstanding = 'yes'`, the value of `shipmentDate` of the corresponding order is not `'today'`.)

Consider materializing a view `V` that is a natural join of those tuples in `Shipments` and `Inventory` where `shipmentDate = 'today'` and `isOutstanding = 'yes'`. Suppose that in the answer to `V` we have (at least) attributes `storeID` and `orderNumber`. Then we can use this view as a *filter* — essentially like an index or a semijoin — to narrow down the number of tuples in the large result of joining the relations `Shipments` and `Inventory`. The query plans that use `V` all join `V` with either `Shipments` or `Inventory` first (an index can be maintained on `V` to make the join efficient), and then join the resulting *smaller-size* temporary relation with the remaining relation in the `FROM` clause of the query.[2]

[2] In presence of frequent updates on the stored data, the update problem for the view `V` is not much more complex than the problem of updating indexes on `Shipments` or `Inventory`, because we have just two relations in the `FROM` clause of the view.

We have seen that under set semantics, having a materialized view such as V could improve the processing efficiency of the daily-report query, which is important when the query is asked often and regularly on large relations.[3] At the same time, the choice of output attributes in a filtering view of this type depends on a number of criteria, including the types of other queries in the query mix. Thus, potentially we would have to consider *all* subsets of the combination of all attributes of the stored relations Shipments and Inventory.

We show in this paper that unlike the case of set semantics, filtering views do not need to be considered in view selection under bag or bag-set semantics. Thus, in view selection we can further restrict the search space of views that are useful in rewriting the given workload queries.

In this paper we present results on designing views to answer queries and on rewriting queries in relational databases under set, bag and bag-set semantics, which are useful in practice. The contributions are the following: (1) We give a bound for the number of subgoals in the optimal viewsets, and (2) we study the computational complexity of the view-selection problem. The results can be used in finding sound and complete algorithms for designing views and for rewriting queries under each of the three semantics.

To the best of our knowledge, limited related work has been done. [7] studies the containment problem of conjunctive queries under bag semantics which is proved to be $\prod_2^p$-hard, whereas equivalence under bag semantics has the same complexity as the graph-isomorphism problem, which is in NP. [10] presents techniques for bag semantics, bag-specific constraints (UWDs), and for handling bag queries over arbitrary mixes of bag and set schema elements and views. The problem of optimizing queries with materialized views under bag semantics is studied in [6] and under set semantics in [18]. Finally, [16] studies conjunctive queries over generalized databases, to obtain an understanding of the behavior of relations as multisets (cf. SQL query-evaluation semantics).

2 Preliminaries

2.1 Basic Definitions

A *relational database* is a collection of stored relations. Each relation R is a collection of tuples, where each tuple is a list of values of the attributes in the *relation schema* of R. A relation can be viewed either as a *set* or as a *bag* (another term is *multiset*) of tuples. A bag can be thought of as a set of elements (we call it the *core-set* of the bag) with multiplicities attached to each element. In a *set-valued database*, all stored relations are sets; in a *bag-valued database*, multiset stored relations are allowed.

In this paper we focus on select-project-join SQL queries with equality comparisons, a.k.a. safe *conjunctive queries*. A conjunctive query is a rule of the form: $Q: \quad ans(\bar{X}) \leftarrow \quad e_1(\bar{X}_1), \ldots, e_n(\bar{X}_n)$, where $e_1, \ldots e_n$ are database relations and $\bar{X}, \bar{X}_1, \ldots, \bar{X}_n$ are vectors of variables. The variables in $\bar{X}$ are called

[3] For instance, WalMart maintains a database with billions of rows in stored relations.

head or *distinguished variables* of Q, whereas the variables in $\bar{X}_i$ are called *body variables* of Q. A query has *self-joins* if the minimized query definition [5] has at least two subgoals with the same relation name. A *view* refers to a named query. A view is said to be *materialized* if its answer is stored in the database.

We say that a bag B is a *subbag* [7] of a bag B' (we write $B \subseteq_b B'$) if each element of B is contained in B' with a greater than or equal multiplicity. The *bag union* ($\sqcup$) [7] of two bags is obtained by adding the multiplicity factors for each tuple in each bag.

2.2 Query Containment and Equivalence

A query Q_1 is *set-contained* in a query Q_2, denoted by $Q_1 \sqsubseteq_s Q_2$, if for any set-valued database $\mathcal{D}$ the answer to Q_1 on $\mathcal{D}$ under set semantics is a subset of the answer to Q_2 on $\mathcal{D}$ under set semantics. A query Q_1 is *bag-contained* (*bag-set contained*) in Q_2, denoted by $Q_1 \sqsubseteq_b Q_2$ ($Q_1 \sqsubseteq_{bs} Q_2$, respectively), if for any bag-valued (set-valued, respectively) database $\mathcal{D}$, the answer to Q_1 on $\mathcal{D}$ under bag semantics (bag-set semantics, respectively) is a subbag of the answer to Q_2 on $\mathcal{D}$ under the same semantics. Two queries are *equivalent under set/bag/bag-set semantics* ($Q_1 \equiv_s Q_2$, $Q_1 \equiv_b Q_2$, $Q_1 \equiv_{bs} Q_2$, respectively) if they are contained in each other under that semantics.

2.3 Equivalent Rewritings and the View-Selection Problem

Let $\mathcal{V}$ be a set of views defined on a database schema $\mathcal{S}$, and $\mathcal{D}$ be a database with the schema $\mathcal{S}$. Then by $\mathcal{D}_\mathcal{V}$ we denote the database obtained by computing all the view relations in $\mathcal{V}$ on $\mathcal{D}$. Let Q be a query defined on $\mathcal{S}$, and $\mathcal{V}$ be a set of views defined on $\mathcal{S}$. A query R is a *rewriting of the query Q using the views in $\mathcal{V}$* if all subgoals of R are view predicates defined in $\mathcal{V}$ or interpreted predicates.

The *expansion* R^{exp} of a rewriting R of a query Q using views is obtained from R by replacing all the view atoms in the body of R by their definitions in terms of the base relations. A rewriting R of a query Q on a set of views $\mathcal{V}$ is a *contained rewriting* of Q using $\mathcal{V}$ under set semantics if for every database $\mathcal{D}$, $R(\mathcal{D}_\mathcal{V}) \subseteq Q(\mathcal{D})$. A rewriting R of a query Q on a set of views $\mathcal{V}$ is an *equivalent rewriting* under set semantics if for every database $\mathcal{D}$, $R(\mathcal{D}_\mathcal{V}) = Q(\mathcal{D})$.

The definition of the notion *contained rewriting* for *bag* or *bag-set* semantics is analogous. The only difference is that we now require that $R(\mathcal{D}_\mathcal{V})$ is a subbag of $Q(\mathcal{D})$. The definition of the notion of *equivalent rewriting* for the *bag* and *bag-set* semantics is the same as above (in this case, however, the symbol '=' stands for bag equality). A conjunctive equivalent rewriting Q' of a conjunctive query Q (under some semantics) is *locally minimal* [18] if we cannot remove any literals from Q' and still retain equivalence to Q (under the same semantics).

Given a set, or *workload*, $\mathcal{Q}$ of queries on stored relations and a database instance, we want to find and precompute offline a set of intermediate results, defined as views (we call this set of views a *viewset*) on these relations. The views can be used to compute the answers to all queries in the workload $\mathcal{Q}$. Our goal

is to design minimal-cost views, that is, views whose use in the rewriting of the queries in $\mathcal{Q}$ minimizes the evaluation cost of these queries. As we assume that the view relations have been precomputed and stored in the database, we do not assume any cost of computing the views. For query-evaluation costs, we use the following *sum-cost model* [8]. The cost of a join of two relations is proportional to the sum of the sizes of the input and output relations.[4] The cost of a query plan is proportional to the sum of costs of all the joins in the plan. (We assume the use of left-linear query plans, where selections are pushed as far as they go and projection is the last operation.) The cost of evaluating a query is the minimum cost of its query plans. The total cost of evaluating a query workload is proportional to the sum of the costs of its queries; the sum can be weighted to reflect the relative frequency or importance of the queries.

In this paper we consider *problem inputs* that are 4-tuples $(\mathcal{S}, \mathcal{Q}, \mathcal{D}, \mathcal{L})$, where $\mathcal{S}$ is a database schema, $\mathcal{Q}$ is a workload of queries defined on $\mathcal{S}$, $\mathcal{D}$ is a database with schema $\mathcal{S}$, and $\mathcal{L}$ is a collection of constraints on sets of materialized views. A problem input $\mathcal{P}$ is said to be *set-oriented* (*bag-oriented*, *bag-set-oriented*, respectively) if we consider set-semantics (bag-semantics, bag-set semantics, respectively) for computing query answers; $\mathcal{P}$ is said to be *conjunctive* if we consider the conjunctive language for queries, views and rewritings.

Some results in this paper are given for a special type of constraints $\mathcal{L}$ on materialized views: In those results, $\mathcal{L}$ is a singleton $\mathcal{L} = \{C\}$, $C \in \mathbf{N}$. The *storage limit* C means that the total size $size(\mathcal{V}(\mathcal{D}))$ of the relations for the views in $\mathcal{V}$ on $\mathcal{D}$ must not exceed C. If the storage limit is sufficiently large then we can materialize all query answers and this is the optimal viewset. The problem becomes interesting when the storage limit is less than that.[5]

Definition 1. *For a given query Q, semantics (set, bag, or bag-set) for evaluating the query on the database, a viewset $\mathcal{V}$, and database $\mathcal{D}$: (1) R is a* candidate rewriting *of Q in terms of $\mathcal{V}$ if R is an equivalent rewriting of Q under the given semantics, and (2) R is an* optimal rewriting *of Q in terms of $\mathcal{V}$ on $\mathcal{D}$ if R is a candidate rewriting that minimizes the cost of computing the answer to Q on $\mathcal{D}_\mathcal{V}$ among all candidate rewritings of Q in terms of $\mathcal{V}$.*

Definition 2. *Let $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, \mathcal{L})$ be a problem input. A set of views $\mathcal{V}$ is said to be an* admissible viewset *for $\mathcal{P}$ if: (1) $\mathcal{V}$ gives equivalent (*candidate*) rewritings of all the queries in $\mathcal{Q}$, (2) for every view $V \in \mathcal{V}$, there exists an equivalent rewriting of a query in $\mathcal{Q}$ that uses V, and (3) $\mathcal{V}$ satisfies the constraints $\mathcal{L}$.*

Definition 3. *For a problem input $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, \mathcal{L})$, an* optimal viewset *is a set of views $\mathcal{V}$ defined on $\mathcal{S}$, such that: (1) $\mathcal{V}$ is an admissible viewset for $\mathcal{P}$, and (2) $\mathcal{V}$ minimizes the total cost of evaluating the queries in $\mathcal{Q}$ on the database $\mathcal{D}_\mathcal{V}$, among all admissible viewsets for $\mathcal{P}$.*

[4] This models faithfully hash joins and index joins.

[5] If the storage limit is too small then there is no viewset that can rewrite all queries.

Definition 4. *For any problem input $\mathcal{P}$, a viewset $\mathcal{V}$ is said to be* nonredundant *for $\mathcal{P}$ if $\mathcal{V}$ is admissible for $\mathcal{P}$ and there is no proper subset $\mathcal{V}'$ of $\mathcal{V}$ such that $\mathcal{V}'$ is also admissible for $\mathcal{P}$.*

In some results of this paper, instead of a database $\mathcal{D}$ in the definition of a problem input, we will use the notion of an *oracle* $\mathcal{O}$. An oracle is supposed to give, instantaneously, exact relation sizes for all views defined on the schema $\mathcal{S}$. In this case a problem input is written as $(\mathcal{S}, \mathcal{Q}, \mathcal{O}, \mathcal{L})$. The notion of an optimal viewset is defined analogously to the case of problem inputs of the form $(\mathcal{S}, \mathcal{Q}, \mathcal{D}, \mathcal{L})$, where $\mathcal{D}$ is a database. The results in the remainder of this paper are given for problem inputs that include a fixed database, but can be extended in a straightforward manner to problem inputs that include an oracle.

2.4 Different Types of Views

There are two types of conjunctive views that can be used in a conjunctive rewriting of a conjunctive query [9]: (1) containment-target views, and (2) filtering views. A conjunctive view V is a *containment-target view* for a conjunctive query Q if there exists a conjunctive rewriting P of Q (P uses V), and there is a containment mapping (for the set-semantics case, or bijective mappings for the bag and bag-set semantics case) from Q to the expansion P^{exp} of P, such that V provides the image of at least one subgoal of Q under the mapping. Intuitively, in a rewriting, a *containment-target view* "covers" at least one query subgoal. Covering all query subgoals is enough to produce a rewriting of the query. A view is a *filtering view* for a query if it is not a containment-target view.

3 Queries Without Self-joins Under Set Semantics

In this section we consider the view-selection problem under *set* semantics. We present the following results:

1. In Section 3.1 we show that for workloads of queries without self-joins there exist optimal viewsets whose view definitions do not have self-joins. Moreover, the view definitions in such viewsets have no more subgoals than any query in the workload.
2. In Section 3.2 we show that for workloads of queries without self-joins there exist optimal viewsets, such that rewriting any workload query does not require self-joins of containment-target views in the viewset.
3. In Section 3.3 we show that the decision version of the view-selection problem is in NP for workloads of queries without self-joins, provided that filtering views are not used in query rewritings.

These results are very useful in designing an algorithm that constructs optimal viewsets, as they provide a bound on the number of atoms in each view in an optimal viewset and ensure that these views do not contain self-joins, provided the workload queries do not contain self-joins.

3.1 View Definitions Without Self-joins

The following theorem holds for queries without self-joins under set semantics.

Theorem 1. *Given a conjunctive set-oriented problem input $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, \mathcal{L})$, where $\mathcal{L}$ represents a single storage limit and all queries in $\mathcal{Q}$ are conjunctive queries without self-joins, if there exists an optimal viewset $\mathcal{V}$ for $\mathcal{P}$ under the storage limit constraint $\mathcal{L}$, then there exists an optimal viewset $\mathcal{V}'$ under $\mathcal{L}$ such that each view in $\mathcal{V}'$ can be defined as a conjunctive query without self-joins.*

The statement of Theorem 1 is that whenever workload queries have no self-joins then there exist optimal viewsets whose view definitions do not have self-joins. The following Corollary 1 goes one step further, by showing that each view in the optimal viewset has no more subgoals than the workload queries.

Corollary 1. *Given a conjunctive set-oriented problem input $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, \mathcal{L})$ where $\mathcal{L}$ represents a single storage-limit constraint and assuming that (1) all queries in $\mathcal{Q}$ are without self-joins, and (2) the number of (relational) subgoals in any query does not exceed an integer n, then if there exists an optimal viewset for $\mathcal{P}$ under $\mathcal{L}$, then there exists an optimal viewset $\mathcal{V}$ of $\mathcal{P}$ under $\mathcal{L}$, such that for every view V in $\mathcal{V}$, the number of subgoals of V is bounded from above by n.*

The optimal viewset stipulated in Corollary 1 may include filtering views alongside containment-target views. Moreover, even an exponential number of filtering views may be necessary under set semantics; see [8]. Note that we cannot strengthen the Corollary 1 to state that under the premises of the corollary there exists an optimal viewset $\mathcal{V}$, such that each view in $\mathcal{V}$ is a subexpression of some query in the input query workload. As a counterexample, consider Example 3.

Example 3. Consider a query workload $\mathcal{Q}$ = { `Q1, Q2` }, where:

```
Q1(X,Y,Z) :- p(X,X), s(X,Y), t(Y,Z).
Q2(X,Y,Z) :- p(X,Y), s(Y,Y), t(Y,Z).
```

Suppose that we are given a database $\mathcal{D}$= { `p(a,a)`, `p(a,b)`, `p(c,c)`, `s(a,a)`, `s(b,b)`, `s(c,b)`, `t(a,d)`, `t(a,f)`, `t(b,g)`, `t(b,h)`} and a set of constraints $\mathcal{L}$ = { L }, where the value of the storage limit $L = 6$ is an upper bound on the sizes of materialized views on $\mathcal{D}$. Consider a view V:

```
V:  v(Z,T,W,U) :- p(Z,T),s(T,W),t(W,U).
```

Note that the view V is not a subexpression of either query in the workload $\mathcal{Q}$. However, $\mathcal{V}$ = { V } is an optimal viewset for the problem input ({ $P(A,B)$, $S(C,D)$, $T(E,F)$ }, { `Q1, Q2` }, $\mathcal{D}$, { L }), and both input queries can be rewritten using the view V:

```
Q1'(X,Y,Z):- v(X,X,Y,Z).
Q2'(X,Y,Z):- v(X,Y,Y,Z).
```

Finally, when queries have self-joins, the number of subgoals of views can be up to a product of the number of subgoals of the queries; see Example 1 in [8].

3.2 Rewritings Without Self-joins

While the results in Section 3.1 refer to the stucture of the views in an optimal viewset, in this section we are interested in the structure of the query rewritings provided that the queries in the workload $\mathcal{Q}$ do not have self-joins. We show that there exists an optimal viewset $\mathcal{V}$, such that rewriting any query in $\mathcal{Q}$ does not require self-joins of containment-target views in $\mathcal{V}$.

Theorem 2. *Given a conjunctive set-oriented problem input $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, \mathcal{L})$ and assuming that the queries in $\mathcal{Q}$ do not have self-joins, if there exists an optimal viewset $\mathcal{V}$ for $\mathcal{P}$ under the storage limit $\mathcal{L}$, then it is possible to rewrite each query in $\mathcal{Q}$ using $\mathcal{V}$, without self-joins of containment-target views.*

The lack of self-joins in queries is an essential condition in Theorem 2, as we can show that otherwise nontrivial self-joins of containment-target views may be required. An analogous result holds for filtering views (cf. Example 1 in [8]).

3.3 Complexity

We now consider the decision version of the view-selection problem, that is, given a set-oriented problem input $\mathcal{P}$ and a positive integer K, the problem is to determine whether there exists an admissible viewset $\mathcal{V}$ for $\mathcal{P}$, such that the cost of evaluating the queries $\mathcal{Q}$ in $\mathcal{P}$ using $\mathcal{V}$ does not exceed K. We show that this problem is in NP for workloads of queries without self-joins, provided that filtering views are not used in query rewritings. We prove the result for the *oracle* version of problem inputs, that is, we show that the size of a witness is polynomial in the size of the following components of the problem input: database schema, query workload, and constraints on the materialized views. This result is stronger than proving that the size of a witness is polynomial in the size of the above components plus the size of an input database, because database schemas, query workloads, and constraints on the materialized views are typically small in size compared to the size of possible databases conforming to the schemas. To prove the main result, we first establish an upper bound on the number of containment-target views in query rewritings.

Lemma 1. *[18] Let Q be a conjunctive query and $\mathcal{V}$ be a set of views, both Q and $\mathcal{V}$ without built-in predicates. If the body of Q has p relational subgoals and Q' is a locally minimal equivalent conjunctive rewriting of Q using $\mathcal{V}$, then Q' has at most p relational subgoals.*

Proposition 1. *For any conjunctive query Q with p relational subgoals and for any locally minimal conjunctive rewriting Q' of Q in terms of views such that $Q' \equiv_s Q$, the number of containment-target views in Q' does not exceed p.*

This result follows from the observation that any locally minimal rewriting does not contain filtering views. From Proposition 1 we obtain Theorem 3:

Theorem 3. *Given an* oracle version *of a conjunctive set-oriented problem input $\mathcal{P}$ whose queries are without self-joins, the decision version of the view-selection problem is in NP, provided that rewritings do not include filtering views.*

Note that if filtering views are allowed in query rewritings, then the view-selection problem under set semantics has an exponential-time lower bound even when none of the workload queries have self-joins; see [8].

4 Queries Under Bag Semantics

In this section we consider the view-selection problem under bag semantics. Before proceeding to the main results of this section, we note that under bag semantics any candidate query rewriting lacks any filtering views, as well as any redundant containment-target views [7]. We now summarize the main results:

1. In Section 4.1 we show that for workloads of queries with or without self-joins, each view definition in any admissible viewset (and thus in any optimal viewset) has no more subgoals than any query in the input workload. As a consequence, for workloads of queries without self-joins each view definition in an admissible viewset can be defined without self-joins.
2. In Section 4.2 we show that for workloads of queries without self-joins and for any admissible viewset, rewriting any query in the workload does not require self-joins of view atoms.
3. In Section 4.3 we show that the decision version of the view-selection problem is in NP for workloads of queries with or without self-joins and for a single storage-limit constraint on materialized views (see also [1]).

Comparing these results with those in Section 3, we conclude that both constructing admissible/optimal viewsets and rewriting queries using views are easier problems under bag semantics than under set semantics.

4.1 Bounded Number of Subgoals

The following lemma holds for workloads of queries without *or with* self-joins under bag semantics and for arbitrary sets of constraints on materialized views.

Lemma 2. *Let $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, \mathcal{L})$ be a conjunctive bag-oriented problem input, where $\mathcal{L}$ is a set of* any *constraints. Let $\mathcal{V}$ be any* admissible *viewset for $\mathcal{P}$, and let Q be any query in $\mathcal{Q}$. Suppose $\mathcal{V}' \subseteq \mathcal{V}$ is the set of all views in the equivalent rewriting R of Q in terms of $\mathcal{V}$. Then the definitions of views $\mathcal{V}'$ in the expansion of R form a partition of the definition of Q.*

The remaining results in Section 4.1 follow trivially from Lemma 2.

Proposition 2. *Given a conjunctive bag-oriented problem input $\mathcal{P}$, let $\mathcal{V}$ be any admissible viewset for $\mathcal{P}$. Then each view in $\mathcal{V}$ has at most n subgoals, where n is the number of subgoals in the longest query in the input workload $\mathcal{Q}$.*

The following theorem sets an upper bound on the number of subgoals in the body of any view definition in any admissible viewset.

Theorem 4. *Let $\mathcal{P}$ be a conjunctive bag-oriented problem input and n be the number of subgoals in the longest query in $\mathcal{Q}$. Then, for any admissible viewset $\mathcal{V}$ for $\mathcal{P}$, each view in $\mathcal{V}$ can be defined using at most n subgoals.*

We can make a more precise statement about the number of subgoals in view definitions for views in admissible viewsets: Let $\mathcal{P}$ be a conjunctive bag-oriented problem input and let V be any view in any *admissible* viewset $\mathcal{V}$ for $\mathcal{P}$. Suppose V is used in rewriting queries $Q_{i_1}, \ldots, Q_{i_k}$ in $\mathcal{Q}$; let m be the number of subgoals in the *shortest* definition among $Q_{i_1}, \ldots, Q_{i_k}$. Then V can be defined using at most m subgoals. In addition, we observe the following. For any conjunctive bag-oriented problem input $\mathcal{P}$ and for an admissible viewset $\mathcal{V}$ for $\mathcal{P}$: If queries in $\mathcal{Q}$ do not have self-joins, then every view in $\mathcal{V}$ can be defined as a conjunctive query without self-joins.

4.2 Rewritings Without Self-joins of Views

Analogously to the case of set semantics, in the case of bag semantics we can show that for problem inputs $\mathcal{P}$ whose query workloads $\mathcal{Q}$ do not have self-joins, and for any admissible viewset $\mathcal{V}$ for $\mathcal{P}$, rewriting any query in $\mathcal{Q}$ does not require self-joins of views in $\mathcal{V}$. The following theorem follows directly from Lemma 2.

Theorem 5. *Let $\mathcal{P}$ be a conjunctive bag-oriented problem input and $\mathcal{V}$ an admissible viewset for $\mathcal{P}$. Assuming that queries in $\mathcal{Q}$ do not have self-joins, then it is possible to rewrite each query in $\mathcal{Q}$ without using self-joins of views in $\mathcal{V}$.*

4.3 Complexity

In this section we show that the decision version of the view-selection problem is in NP for a single storage-limit constraint on materialized views (see also [1]). We define the decision version of the problem and state the result for the *oracle* version of problem inputs, analogously to the respective formulations in Section 3.3. At the same time, unlike the results in Section 3.3, the NP results for bag semantics hold for workloads of queries without *or with* self-joins.

We first establish an analog of Proposition 1 in Section 3.3:

Proposition 3. *For any conjunctive query Q with p relational subgoals and for any conjunctive rewriting Q' of Q in terms of views, such that $Q' \equiv_b Q$, the number of views in Q' does not exceed p.*

This result follows immediately from the fact that for any equivalent (under bag semantics) rewriting to a query, the rewriting does not contain filtering views or "unnecessary" containment-target views, and is thus locally minimal.

We now establish Theorem 6, which is a direct consequence of the following observation: Under bag semantics, for any problem input $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, \mathcal{L})$ with any set of constraints $\mathcal{L}$, and for any admissible viewset $\mathcal{V}$ for $\mathcal{P}$, the number of views in $\mathcal{V}$ does not exceed p, where p is the total number of relational subgoals in all the queries in the query workload $\mathcal{Q}$ in $\mathcal{P}$.

Theorem 6. *Given an* oracle version *of a conjunctive, bag-oriented problem input $\mathcal{P}$ and assuming that the input set of constraints $\mathcal{L}$ represents a single storage limit, the decision version of the view-selection problem is in NP.*

5 Queries Under Bag-Set Semantics

In this section we consider the view-selection problem under bag-set semantics. Note that all queries mentioned in the results below may have self-joins. In the extended version of the paper we show that filtering views are not needed under bag-set semantics. The main results of this section are:

1. The results in Section 5.1 are similar to those in Section 4.1, which concerns bag semantics. That is, we show that for workloads of queries with or without self-joins, each view definition in any admissible viewset (and thus in any optimal viewset) has no more subgoals than any query in the input workload. As a consequence, for workloads of queries *without* self-joins, each view definition in an admissible viewset can be defined without self-joins. Moreover, we can show that for workloads of queries without self-joins and for any admissible viewsets, rewriting any query in the workload does not require self-joins of view atoms.
2. In Section 5.2 we show that the decision version of the view-selection problem is in NP for workloads of queries with or without self-joins and for a single storage-limit constraint on materialized views.

5.1 Bounded Number of Subgoals

Our first result holds for workloads of queries without *or with* self-joins under bag-set semantics and for arbitrary sets of constraints on materialized views.

Lemma 3. *Let $\mathcal{P}$ be a conjunctive, bag-set-oriented problem input, and let $\mathcal{V}$ be any* admissible *viewset for $\mathcal{P}$. Then each view in $\mathcal{V}$ has at most n subgoals, where n is the number of subgoals in the longest query in the input workload $\mathcal{Q}$.*

All remaining results in this subsection follow directly from Lemma 3.

Theorem 7. *Let $\mathcal{P}$ be a conjunctive bag-set-oriented problem input, and n be the number of subgoals in the longest query in $\mathcal{Q}$. Then, for all admissible viewsets $\mathcal{V}$ for $\mathcal{P}$, each view in each $\mathcal{V}$ can be defined using at most n subgoals.*

We can make a more precise statement about the number of subgoals in view definitions for views in admissible viewsets:

Corollary 2. *Let $\mathcal{P}$ be a conjunctive bag-set-oriented problem input, and let V be any view in any* admissible *viewset $\mathcal{V}$ for $\mathcal{P}$. Suppose V is used in rewriting queries $Q_{i_1}, \ldots, Q_{i_k}$ in $\mathcal{Q}$; let m be the number of subgoals in the* shortest *definition among the definitions of $Q_{i_1}, \ldots, Q_{i_k}$. Then V can be defined using at most m subgoals.*

We also make the following observation: Let $\mathcal{P}$ be a conjunctive bag-set-oriented problem input, and $\mathcal{V}$ an admissible viewset for $\mathcal{P}$. If queries in $\mathcal{Q}$ do not have self-joins, then each view in $\mathcal{V}$ can be defined as a conjunctive query without self-joins. In addition, we can show that for problem inputs $\mathcal{P}$ whose query workloads $\mathcal{Q}$ do not have self-joins and for any admissible viewset $\mathcal{V}$ for $\mathcal{P}$, rewriting any query in $\mathcal{Q}$ does not require self-joins of views in $\mathcal{V}$.

5.2 Complexity

We now show that the decision version of the view-selection problem is in NP for a single storage-limit constraint on materialized views. We formulate the decision version of the problem and state the result for the *oracle* version of problem inputs, similarly to the previous sections. The NP results hold for workloads of queries without *or with* self-joins.

Proposition 4. *For any conjunctive query Q with p relational subgoals and for any conjunctive locally minimal rewriting Q' of Q in terms of views, such that $Q' \equiv_{bs} Q$, the number of views in Q' does not exceed p.*

This result follows from the definition of a locally minimal rewriting that is equivalent to a query under bag-set semantics. By definition, the rewriting does not contain filtering views or "unnecessary" containment-target views.

We now establish that the decision version of the view-selection problem is in NP. This result is a consequence of the following observation: Under bag-set semantics, for any problem input $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, \mathcal{L})$ with any set of constraints $\mathcal{L}$, and for any nonredundant viewset $\mathcal{V}$ for $\mathcal{P}$, the number of views in $\mathcal{V}$ does not exceed p, where p is the total number of relational subgoals in all the queries in the query workload $\mathcal{Q}$ in $\mathcal{P}$.

Theorem 8. *Given an* oracle version *of a conjunctive bag-set-oriented problem input $\mathcal{P}$ and assuming that the input set of constraints $\mathcal{L}$ represents a single storage limit, the decision version of the view-selection problem is in NP.*

6 Conclusions and Future Work

This paper presents results on designing views to answer queries in relational databases under set, bag and bag-set semantics. The results can be used in finding sound and complete algorithms for designing views and rewriting queries under each of the three assumptions. We are currently working on designing such algorithms. In our future work we also plan to study the complexity of the optimization problem and to extend this method to include, in a systematic way, queries with arithmetic comparisons. Applying these results to XQuery is another direction of our future research.

References

1. F. Afrati and R. Chirkova. Selecting and using views to compute aggregate queries. In *Proceedings of ICDT*, 2005.
2. F. Afrati, C. Li, and J.D. Ullman. Generating efficient plans for queries using views. In *Proceedings of ACM SIGMOD*, 2001.
3. S. Agrawal, S. Chaudhuri, and V.R. Narasayya. Automated selection of materialized views and indexes in SQL databases. In *Proc. VLDB*, pages 496–505, 2000.
4. E. Baralis, S. Paraboschi, and E. Teniente. Materialized view selection in a multidimensional database. In *Proceedings of VLDB*, pages 156–165, 1997.

5. A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proc. 9th ACM STOC*, pages 77–90, 1977.
6. S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *Proceedings of ICDE*, pages 190–200, 1995.
7. S. Chaudhuri and M. Y. Vardi. Optimization of real conjunctive queries. In *Proceedings of PODS*, pages 59–70. ACM Press, 1993.
8. R. Chirkova, A. Y. Halevy, and D. Suciu. A formal perspective on the view selection problem. *The VLDB Journal*, 11(3):216–237, 2002.
9. R. Chirkova and C. Li. Materializing views with minimal size to answer queries. In *Proceedings of PODS*, pages 38–48. ACM Press, 2003.
10. A. Deutsch. *XML Query Reformulation over Mixed and Redundant Storage*. PhD thesis, University of Pennsylvania, 2002.
11. S. Gribble, A. Halevy, Z. Ives, M. Rodrig, and D. Suciu. What can databases do for peer-to-peer? In *Proceedings of WebDB*, 2001.
12. H. Gupta. Selection of views to materialize in a data warehouse. In *Proceedings of ICDT*, pages 98–112, 1997.
13. H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman. Index selection for OLAP. In *Proceedings of ICDE*, pages 208–219, 1997.
14. H. Gupta and I. S. Mumick. Selection of views to materialize under a maintenance cost constraint. In *Proceedings of ICDT*, pages 453–470, 1999.
15. V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proceedings of ACM SIGMOD*, pages 205–216, 1996.
16. Y. Ioannidis and R. Ramakrishnan. Containment of conjunctive queries: Beyond relations as sets. *ACM Transactions on Database Systems*, 20(3):288–324, 1995.
17. H. J. Karloff and M. Mihail. On the complexity of the view-selection problem. In *Proceedings of PODS*, pages 167–173, Philadelphia, Pennsylvania, 1999.
18. A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proceedings of PODS*, pages 95–104. ACM Press, 1995.
19. R. Pottinger and A. Y. Levy. A scalable algorithm for answering queries using views. In *Proceedings of VLDB*, pages 484–495, 2000.
20. D. Theodoratos and T. Sellis. Data warehouse configuration. In *Proceedings of VLDB*, pages 126–135, Athens, Greece, 1997.
21. J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. In *Proceedings of VLDB*, pages 136–145, 1997.

The Multi-depot Periodic Vehicle Routing Problem

Aristide Mingozzi

Department of Mathematics, University of Bologna, Bologna, Italy
mingozzi@csr.unibo.it

The Multi-Depot Periodic Vehicle Routing Problem (MDPVRP) is the problem of designing, for an homogeneous fleet of vehicles of capacity Q, a set of routes for each day of a given p-day period. The routes of day k must be executed by m_k vehicles based at the depot assigned to day k. Each vehicle performs only one route per day and each vehicle route must start and finish at the same depot. Each customer i may require to be visited on f_i (say) different days during the period and these visits may only occur in one of a given number of allowable day-combinations. For example, a customer may require to be visited twice during a 5-day period imposing that these visits should take place on Monday-Thursday or Monday-Friday or Tuesday-Friday. The MDPVRP consists of simultaneously assigning a day-combination to each customer and designing the vehicle routes for each day of the planning period so that each customer is visited the required number of times, the number of routes on each day does not exceed the number of vehicles available at the depot assigned to that day and the total cost of the routes is minimized.

The MDPVRP considered in this paper generalizes three well known routing problems: the Periodic Vehicle Routing Problem (PVRP), the Multi-Depot Vehicle Routing Problem (MDVRP) and the single depot Capacitated Vehicle Routing Problem (CVRP):

The PVRP and the MDVRP are special cases of the MDPVRP.

The MDPVRP becomes the PVRP when all vehicles are located at one depot (i.e. nd=1) but only m_k of these vehicles can be used on day k.

The MDVRP is a special case of the MDPVRP as it is defined on a single day (i.e. p=1) but vehicles are located on several depots and each vehicle route must start and finish at the same depot. Moreover, each customer must be visited exactly once by one of the vehicles located at the depots specified by the customer.

The MDPVRP becomes the single depot CVRP when the planning period is of one day only (i.e. p=1) and every customer must be visited only once (i.e. f_i=1, for every customer i) from a single depot (i.e. nd=1) where all vehicles are located.

It is quite obvious that the CVRP is also a special case of both the PVRP and the MDVRP.

At our knowledge the MDPVRP has not be considered in the literature. Despite their practical interest, only few papers are reported for the PVRP and the MDVRP.

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 347–350, 2005.

The PVRP has many practical applications in the grocery industry [5], the soft drink industry (vending machines), the automotive industry (parts distribution), industrial gases distribution and refuse collection ([4, 21]).

All papers on the PVRP reported in the literature present heuristic methods whose quality is unknown since no optimal solutions or lower bounds are provided.

Early heuristics were proposed by [4, 21]. More sophisticated heuristic algorithms were presented by [8, 22, 20, 7, 11, 9]. The tabu search proposed by [9] is capable of solving the PVRP as well as the Periodic Travelling Salesman Problem (PTSP) and the MDVRP. The computational results indicate that this method outperforms all existing heuristics for all three problems. To our knowledge no exact method was proposed in the literature for the PVRP.

At our knowledge the only exact methods published for the MDVRP are due to [14, 15]. Heuristic algorithms for the MDVRP were proposed by [24, 23, 27, 12, 6, 19].

The CVRP has been extensively investigated in the literature (see the surveys of [26, 17]). Recently, new effective exact algorithms for the CVRP have been proposed by [1, 16, 3, 10, 2]. This latter method was developed for solving Time constrained Vehicle Routing Problem on a directed MultiGraph (TVRP-MG) and is based on the methods for the CVRP proposed by Mingozzi, Christofides and Hadjicostantinou (1994) and by [13].

The procedures of [10] and of [2] are both based on the Set Partitioning (SP) formulation of the CVRP and solved problem E-n76-k10, the most famous CVRP instance that until recently was unsolved.

The methods of [1, 16, 3] are branch and cut algorithms capable of solving to optimality the largest instance of the symmetric CVRP (135 customers) reported in the literature but cannot solve problem E-n76-k10.

In this paper, we describe an integer programming formulation (F) of the MDPVRP that is an extension of Set Partitioning formulation of the CVRP. Problem F cannot be solved directly, even for moderate size instances, as the number of variables is as large as the total number of routes that is typically exponential.

We describe an exact method for solving problem F that uses variable pricing in order to reduce the set of variables to solvable proportions. The pricing method is based on a bounding procedure for finding near optimal solutions of the dual problem of the LP relaxation of problem F, called problem D.

The bounding method is an additive procedure that computes a lower bound on the MDPVRP as the sum of the dual solution costs obtained by a sequence of five different heuristics for solving D, where each heuristic explores a different structure of the MDPVRP. Three of these procedures are based on relaxations that do not require the generation of the route set. The other two procedures combine subgradient optimization with column generation to produce a lower bound of the same quality of the value of the LP relaxation of formulation F without being affected by the typical degeneration of the classical column generation methods based on simplex LP solvers.

The exact procedure uses the dual solution to generate a limited set of feasible routes so that the resulting reduced problem F can be solved by an integer programming solver. If optimality is not achieved, then the procedure is iteratively repeated with a larger subset of routes until either optimality is proved or the distance from optimality is below an *a priori* defined threshold level.

Computational results on test problems derived from the literature are reported for the PVRP, the MDVRP and the CVRP.

References

1. Augerat, P., J. M. Belenguer, E. Benavent, A. Corberan, D. Naddef, G. Rinaldi: Computational results with a branch and cut code for the capacitated vehicle routing problem. Rapport de recherche 1 RR949-M, ARTEMIS-IMAG, Grenoble France (1995).
2. Baldacci, R., L. D. Bodin, A. Mingozzi. The Multiple Disposal Facilities and Multiple Inventory Locations Rollon-Rolloff Vehicle Routing Problem. Computers and Operations Research (to appear).
3. Baldacci, R., E. Hadjiconstantinou, A. Mingozzi: An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. Oper. Res. 52 (2004) 723-738.
4. Beltrami, E. J., L. D. Bodin: Networks and vehicle routing for municipal waste collection, Networks 4 (1974) 65-94.
5. Carter, M. W., J.M. Farolden, G. Laporte, J. Xu: Solving an integrated logistics problem arising in grocery distribution. INFOR 34 (1996) 290-306.
6. Chao, I. M., B. L. Golden, E. A. Wasil: A new heuristic for the multi-depot vehicle routing problem that improves upon best-known results. Am .J. Math. Mgmt. Sci. 13 (1993) 371-406.
7. Chao, I. M., B. L. Golden, E. A. Wasil: An improved heuristic for the period vehicle routing problem. Networks 26 (1995) 22-44.
8. Christofides, N., J. E. Beasley: The period routing problem, Networks 14 (1984) 237-256.
9. Cordeau, J. F. M. Gendreau, G. Laporte: A tabu search heuristic for periodic and multi-depot vehicle routing problems, Networks 30 (1997) 105-119.
10. Fukasawa, R, J. Lysgaard, M. Poggi de Aragao, M. Reis, E. Uchoa, R. F. Werneck: Robust branch-and-cut-and-price for the capacitated vehicle routing problem. In G. Nemhauser and D. Bienstock (eds.) Integer Programming and Combinatorial Optimization 10. Lecture Notes in Computer Science, Vol. 3064, Springer-Verlag (2004) 1-15.
11. Gaudioso, M., G. Paletta: A heuristic for the periodic vehicle routing problem. Trans. Sci. 26 (1992) 86-92.
12. Gillet, B. E., J. G. Johnson: Multi-terminal vehicle-dispatch algorithm. Omega 4 (1976) 711-718.
13. Hadjiconstantinou, E., N. Christofides, A. Mingozzi: A new exact algorithm for the vehicle routing problem based on q-paths and K-shortest paths relaxations. Annals of Operations Research 61 (1995) 21-43.
14. Laporte, G., Y. Nobert, D. Arpin: Optimal solutions to capacitated multi-depot vehicle routing problem. Congress. Num. 44 (1984) 283-292.
15. Laporte, G., Y. Nobert, S. Taillefer: Solving a family of multi-depot vehicle routing and location-routing problems. Trans. Sci. 22 (1988) 161-172.

16. Lysgaard, J., A.N. Letchford, R.W. Eglese: A new branch-and-cut algorithm for the capacitated vehicle routing problems. Mathematical Programming 100 (2004) 423-445.
17. Naddef, D., G. Rinaldi: Branch-and cut algorithms for the capacitated VRP. In The Vehicle Routing Problem, P. Toth and D. Vigo (eds.). SIAM, Philadelphia (2002).
18. Raft, O. M.: A modular algorithm for an extended vehicle scheduling problem. Eur. J. Oper. Res. 11 (1982) 67-76.
19. Renaud, J., G. Laporte, F.F. Boctor: A tabu search heuristic for the multi-depot vehicle routing problem. Comput. Oper. Res. 23 (1996) 229-235.
20. Russel, R. A., D. Gribbin: A multiphase approach to the period routing problem. Networks 21 (1991) 747-765.
21. Russel, R. A., W. Igo: An assignment routing problem. Networks 9 (1979) 1-17.
22. Tan, C.C.R., J.E. Beasley: A heuristic algorithm for the period routing problem. Omega 12 (1984) 497-504.
23. Tillman, F.A., T. M. Cain: An upper bound algorithm for the single and multiple terminal delivery problem. Mgmt. Sci. 18 (1972) 664-682.
24. Tillman, F.A., R.W. Hering: A study for look-ahead procedure for solving the multiterminal delivery problem. Trans. Res. 5 (1971) 225-229.
25. Toth, P., D. Vigo. The vehicle routing problem. SIAM, Philadelphia, Monographs on Discrete Mathematics and Applications, Philadelphia (2002).
26. Toth, P., D. Vigo: Branch-and-bound algorithms for the capacitated VRP. In The vehicle routing problem. Toth P. and Vigo D. (eds.), SIAM Monographs on Discrete Mathematics and Applications, Philadelphia (2002).
27. Wren, A., A. Holliday: Computer scheduling of vehicles from one or more depots to a number of delivery points. Oper. Res. Q. 23 (1972) 333-344.

Abstract Representation in Painting and Computing

Robert Zimmer

Goldsmiths Digital Studios, Goldsmiths College University of London,
New Cross, London SE14 6NW, UK
R.Zimmer@gold.ac.uk

This paper brings together two strands of my research: an interest in abstraction in AI computing systems (see for example [1]) and an interest in the study of paintings as a key to understanding perception and cognition (see, for example, [2]). Our senses of the world are informed by the art we make and by the art we inherit and value, works that in them selves encode others' worldviews. This two-way effect is deeply rooted and art encodes and affects both a culture's ways of perceiving the world and its ways of remaking the world it perceives. The purpose of this paper is to indicate ways in which a study of abstraction in art can be used to discover insights into our perception of the world and how these insights may be employed, in turn, to develop computing systems that can take advantage of some of these forms of abstraction both in their own processing and in the way they present themselves to users.

References

1. Holte, R.C., Mkadmi, T., Zimmer, R.M., MacDonald, A.J.: Speeding Up Problem-Solving by Abstraction: A Graph-Oriented Approach. Artificial Intelligence, vol. 85, (1996) 321–361
2. Zimmer, R.: Abstraction in Art with Implications for Perception. Philosophical Transactions of the Royal Society B (June 2003)

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, p. 351, 2005.

Categorizing Gene Expression Correlations with Bioclinical Data: An Abstraction Based Approach

Arriel Benis

LIM&Bio, Université Paris 13 – Paris Nord, Bobigny, France
benis@limbio-paris13.org

Our research takes place in a bioinformatics team embedded in a biological unit where the biologists are using pangenomics cDNA chips to measure expression level of thousands of genes at a time. The goal of our research is to systematically categorize of relations between genes expression levels (1) and biomedical values to support finding of candidate genes allowing a better diagnostic of obesities and related diseases (2). A key issue in the analysis of cDNA chips is that the number of expression levels per chip is very high compared to the number of chips. We are working with 40 cDNA chips with ±40000 spots each one and with 2 biomedical parameters. One way used by biologists to discover relationships between these types of data consists in computing correlations for a small number of them based on their biological knowledge. To go beyond such a biased and manual selection, we propose to explore automatically combinations between all available bioclinical parameters with all gene expressions. These new data need to be classify to identify significant Linear Correlation Discoveries (3). Our method, DISCOCLINI, consists in using abstraction operators to remove outliers, approximation to define correlations and reformulation to describe and to cluster correlations by variations patterns.

Biomedical data are subject to high variability due to manipulation errors, interindividual variability, etc. This induces the presence of noise and particularly of outliers in data. These ones are isolated values of a given gene expression or a clinical parameter. A first step consists in discovering significantly correlated genes with bioclinical parameters. One computational problem consists in detecting and filtering (4) the outliers. We define two ways to detect the outliers. The first one, OUTTRIM, is based on the trimming of points which consists in cutting down x% of the extreme values in the sets. As this approach is very stringent, we explored, another method based on PAM (Partitioning Around Medoids) (5), OUTPAM. This one consists in grouping data points in a predefined number of clusters. In this context an outlier is define as a cluster reduced to a single point. Thanks to PAM, we could detect more relevant outliers than with the first approach. Once outliers are removed from the data, global and local Spearman correlation rank between gene expression data and bioclinical parameters may be computed. This step generates results which need to be explored and to be compared for a given support (in number of individuals).

To visualize the previous results we propose a colored language of symbols. This language, noted L1, allows to represent 2D plot in a gradient colored 1D plot. This allows easily visualization and comparison found relations (see fig.1). As we are only

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 352 – 353, 2005.

interested in the variation profiles and not in the variation intensities is reformulated in a language, L2 based on six symbols: "↗" for a positive relation, "↘" for a negative one, "→" for a null, "▪" for an outlier, "⤒" and "⤓" for breaking intervals. All the relations are redefined in L2 and then classified in a Galois lattice to associate a variation pattern to a biset distribution. Each node of this space corresponds to a cluster of specific relations detected. In a final step, the most *relevant* clusters are studied in a biomedical point of view. This phase allows the user to evaluate and to validate L2, the lattice and it description of patterns.

Preliminary results of DISCOCLINI processing have supported the discoveries of a set of genes regulated by Epinephrine in human muscle (6) and of Cathepsin S, as novel biomaker of adiposity (7).

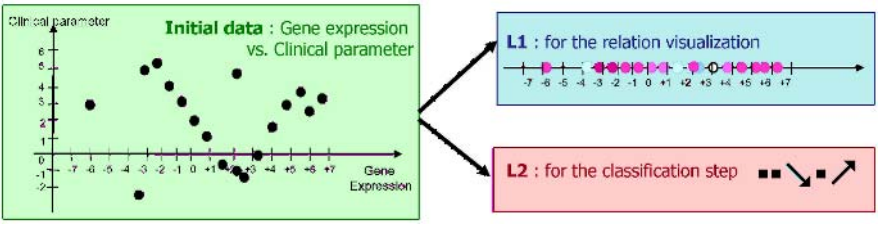


Fig. 1. Example of reformulation languages

We acknowledge the Benjamin Delessert Institute for its financial support.

References

1. Schena, M., Shalon, D., Davis, R. W., and Brown, P. O. (1995) Quantitative monitoring of gene expression patterns with a complementary DNA microarray. *Science* 270, 467-470
2. Clement, K., Boutin, P., and Froguel, P. (2002) Genetics of obesity. *Am J Pharmacogenomics* 2, 177-187
3. Roger H.L. Chianga, Cecilb, C. E. H., and Limc, E.-P. (2005) Linear correlation discovery in databases: a data mining approach. *Data and Knowledge Engineering* 53, 311-337
4. Zucker, J. D. (2003) A grounded theory of abstraction in artificial intelligence. *Philos Trans R Soc Lond B Biol Sci* 358, 1293-1309
5. Kaufman, L., and Rousseeuw, P. (1990) *Finding Groups in Data: An Introduction to Cluster Analysis*, Wiley, New York
6. Viguerie, N., Clement, K., Barbe, P., Courtine, M., Benis, A., Larrouy, D., Hanczar, B., Pelloux, V., Poitou, C., Khalfallah, Y., Barsh, G. S., Thalamas, C., Zucker, J. D., and Langin, D. (2004) In vivo epinephrine-mediated regulation of gene expression in human skeletal muscle. *J Clin Endocrinol Metab* 89, 2000-2014
7. Taleb S, Lacasa D, Bastard JP, Poitou C, Cancello R, Pelloux V, Viguerie N, Benis A, Zucker JD, Bouillot JL, Coussieu C, Basdevant A, Langin D, and K, C. (2005, in press) Cathepsin S, a novel biomaker of adiposity : relevance to atherogenisis. *FASEB Journal*

Learning Abstract Scheduling Models*

Tom Carchrae[1] and J. Christopher Beck[2]

[1] Cork Constraint Computation Centre, University College Cork, Ireland
t.carchrae@4c.ucc.ie
[2] Department of Mechanical & Industrial Engineering, University of Toronto, Canada
jcb@mie.utoronto.ca

1 Introduction

In practice, most scheduling problems are an abstraction of the real problem being solved. For example, when you plan your day, you schedule the activities which are critical; that is you schedule the activities which are essential to the success of your day. So you may plan what time to leave the house to get to work, when to have meetings, how you share your vehicle with your spouse and so on. On the other hand, you probably do not consider the activities that are easy to arrange like brushing your teeth, going to the shops, making photocopies and other such tasks that can usually be accomplished whenever you have the time available. Often, if a schedule goes wrong, it is because a missed or underestimated activity had a significant impact on the schedule. We typically learn which activities are critical by experience and create an abstract scheduling problem including only these critical activities. Instead of scheduling the non-critical activities we estimate their effects in the abstract scheduling problem.

Most industrial scheduling problems are also an abstraction of the real problem. It is common to only schedule activities on bottleneck resources and let the other activities fall into place once these are scheduled. Deciding which resources are bottlenecks is currently done by experienced human experts who determine what needs to be considered to create an accurate and high quality schedule. Unfortunately this process is both time consuming and error prone. We want to automate the process of creating a good quality abstract scheduling model.

2 Learning Abstract Scheduling Models

We consider the following system. A series of scheduling problems are presented to a system, one after another, perhaps on a weekly or daily basis. The objective of the system is to produce a high quality schedule within a limited amount of processing time. The system chooses an abstract model and searches for good solutions. Once a good solution has been found it is extended to create a solution to the full problem.

* This work has received support from Science Foundation Ireland under Grant 00/PI.1/C075, Irish Research Council for Science, Engineering, and Technology under Grant SC/2003/82, and ILOG, SA.

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 354–355, 2005.

For example, say we choose to schedule only the activities on the 10 busiest machines in the factory. We create a high quality schedule for these activities. Then we extend this abstract solution into a full schedule for all of the activities by inserting the remaining activities into the schedule.

Once the system has produced a schedule a human user can examine the solution. If they are satisfied with the result, they can release the schedule for execution in the factory. However, if they are not happy with the solution, they can modify the abstract model by changing the critical activities. They might want to make changes because important jobs are not completed soon enough or an expensive machine is not being utilised as efficiently as they would like.

While the system has to produce schedules quickly for each problem instance, it is allowed a longer time between instances to analyse how it could have performed better. During this time, it will explore different approaches. Upon receiving the next problem instance it can then draw on its previous experience, including the human feedback on its abstract models, to choose a better abstract scheduling model.

3 Research Aims

There are several challenging research questions which must be explored to build a scheduling system that learns a good abstract scheduling model:

- We need to determine how to learn a good abstract model for a particular factory. A good abstract model will produce solutions which have a strong correlation to good solutions to the full problem. Initially, we can use heuristics to analyse the problem instance and determine which activities are most likely to compete for a resource. However, as the system gains experience in scheduling the problem, we can learn by exploring different abstract models and evaluate them by extending them to a full solution. Our goal is to learn a good abstract model that will consistently produce good quality schedules to the full problem.
- How do we approximate the missing parts of an abstract model? In particular, when we choose to schedule only the critical activities, what do we do with the non-critical activities. We do not wish to ignore them as if they do not exist, as ultimately they must be considered in the full solution. One approach is to approximate their effects using precedence constraints with a minimum delay in the abstract model. These precedence constraints reserve gaps in the schedule and do not require any search effort. Determining the size of these gaps is an interesting research question.
- When we extend an abstract solution into a full solution there are several approaches. We can fix the sequence of activities in the abstract solution and schedule the remaining activities subject to this sequence. Another approach is to use the abstract solution to guide a heuristic to construct a full schedule or we can use the abstract solution as a starting point for the full solution and allow decisions to be changed while searching for a full solution, eg. using local search.

Knowledge Acquisition on Manipulation of Flow and Water Quality Models

Kwok Wing Chau

Dept. of Civil & Structural Engineering, Hong Kong Polytechnic University, Hunghom, Kowloon, Hong Kong
cekwchau@polyu.edu.hk

Abstract. At present, numerical simulation has become the norm to mimic coastal flow and water quality problems. Yet, results from algorithmic procedure sometimes appear unreasonable, in particular when a model is initially set up. The incorporation of existing heuristic knowledge on model manipulation into the modeling system is able to provide great assistance to novice users who may lack the requisite knowledge to establish the model properly. Knowledge comprising the correct manipulation direction and the means to enhance manipulation effectiveness is encapsulated. Through an appropriate knowledge acquisition facility, recent artificial intelligence technology is coupled into numerical modeling system for simulating the manipulation process. It is able to assist the user to formulate a suitable strategy for striking a balance between accuracy and effectiveness and to tune the model to accomplish satisfactory modeling of real phenomena. It can bridge the existing gap between numerical modelers and practitioners in this domain.

Research Summary

Modeling can be considered as a process that transforms knowledge regarding real phenomena into numerical formats, simulates for the behaviors, and translates the numerical results back to comprehensible formats. It is an interaction between knowledge and information in the form of knowledge to information and then back to knowledge again. Over these years, a myriad of numerical models has become available for engineering problems. They are increasingly comprehensive and include a diversity of coastal processes. Since the emphasis has been traditionally concentrated on algorithmic procedures, models are often not user-friendly and lack knowledge transfers in model interpretation. This produces substantial constraints on model uses and considerable gaps between model developers and practitioners.

Selection and manipulation of an appropriate numerical model to solve a practical problem is a specialized task, entailing detailed knowledge of the applications and limitations of the model. Many model users do not possess the requisite knowledge to glean their input data, build algorithmic models and evaluate their results. This may produce inferior design and cause under-utilization, or even total failure, of these models. Model manipulation is often an inevitable process, in particular during the

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 356 – 357, 2005.

initial model establishment. The process is iterated for several cycles until the results meet the threshold of mathematical and physical correctness preset by the user. It can be regarded completed when satisfactory simulation of real phenomena is attained. In general, the process on model manipulation is time-consuming and depends highly on the experience of the modeler. Expertise knowledge is required for selecting a suitable model to apply in a specific situation. A well-experienced modeler may use the heuristic knowledge unconsciously to undertake model manipulation. It is considered that the incorporation of existing heuristic knowledge on model manipulation into the modeling system is able to provide great assistance to novice users.

Some previous efforts have been made to support a wider scope of model users, through a variety of means including menu of parameter specification, automatic grid formation, pre-processing and post-processing facilities. These tools act as intelligent front-ends in handling models for specific flow or water quality problems. Nevertheless, they do not address the core problem of knowledge elicitation and transfer. Successful applications of expert system technology have also been reported in the selection of numerical model in coastal engineering [1-9]. However, they are limited to one-dimensional modeling systems, and represent only a minute portion of knowledge in this field. Moreover, their knowledge bases include heuristic rules for model selection but not manipulation. Literature on the incorporation of expert system technology into model manipulation is still scarce to date.

Hence, the principal objective of this work is to integrate recent artificial intelligence technology for model manipulation for flow and water quality. Through establishing a befitting knowledge acquisition facility, the system has immense potential in facilitating non-experienced user in both model selection and manipulation.

List of Relevant Publications

1. Chau, K.W.,: Selection and Calibration of Numerical Modeling in Flow and Water Quality Modeling. Environmental Modeling and Assessment 9(3) (2004) 169-178
2. Chau, K.W., Chen, W.: An Example of Expert System on Numerical Modelling System in Coastal Processes. Advances in Engineering Software 32(9) (2001) 695-703
3. Chau, K.W., Chen, W.: A Fifth Generation Numerical Modeling System in Coastal Zone. Applied Mathematical Modelling 25(10) (2001) 887-900
4. Chau, K.W., Cheng, C.T., Li, C.W.: Knowledge Management System on Flow and Water Quality Modeling. Expert Systems With Applications 22(4) (2002) 321-330
5. Chau, K.W., Yang, W.W.: A Knowledge-Based Expert System for Unsteady Open Channel Flow. Engineering Applications of Artificial Intelligence 5(5) (1992) 425-430
6. Chau, K.W., Yang, W.W.: Development of an Integrated Expert System for Fluvial Hydrodynamics. Advances in Engineering Software 17(3) (1993) 165-172
7. Chau, K.W., Yang, W.W.: Structuring and Evaluation of VP-Expert Based Knowledge Bases. Engineering Applications of Artificial Intelligence 7(4) (1994) 447-454
8. Chau, K.W., Yang, W.W.: Knowledge Acquisition and Representation for Unsteady Open Channel Flow. Journal of Intelligent Systems 6(3-4) (1996) 221-237
9. Chau, K.W., Zhang, X.N.: An Expert System for Flow Routing in a River Network. Advances in Engineering Software 22(3) (1995) 139-146

Abstraction and Multiple Abstraction in the Symbolic Modeling of the Environment of Mobile Robots

Juan-Antonio Fernandez-Madrigal, Javier Gonzalez, and Cipriano Galindo

Dpto. Ingenieria de Sistemas y Automatica, E.T.S.I. Informtica,
Universidad de Malaga, 29071 Malaga, Spain
{jafma, jgonzalez, cipriano}@ctima.uma.es

Except for pure reactive robots, that do not work with any explicit representation of their world [1], an intelligent robot must possess some symbolic representation of its environment in order to reason, plan (prediction), and perform efficiently (due to the intractable amount of subsymbolic information acquired from the real world). We have been working on that area during the last decade, in particular exploring the advantages of using abstraction and multiple abstraction for modeling the environment of a mobile robot. In this sense we have addressed the following main issues:

1. *Automatic construction of the model.* Assistive approaches (involving human operators) are possible [4] but limit the autonomy of the robot.
2. *Automatic optimization* (adaptation) of the model, for coping with the different situations that the robot may face during its operation without constructing an entirely new model each time from scratch.
3. *Coherence* between the symbols in the model and the real world. This must be addressed as a dynamic procedure since the real world changes continuously.
4. *Efficiency* in using the model. We believe that the best model for a given robot is the one that improves the most the robot's planning of operations.

Up to now, we have been working on obtaining a comprehensive solution with abstraction as a basis for coping with all these issues at once. In the next we describe in more detail our solutions to each one[1].

For coping with point 4), we use a symbolic model of the environment that represents explicitly relational information (for example: places to navigate related by navigational paths), and also abstraction, based on mathematical graphs. We have two "flavors" for that model: one that adds simple or conventional abstraction to the graphs and another one that enriches it with multiple abstraction. Simple abstraction of detail consists of stacking several graphs that represent the same information with different levels of detail, which has demonstrated important improvements in computational efficiency for a number of

[1] We follow the chronological order in which we addressed the points above.

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 358–359, 2005.

operations [2],[5],[6],[7]. We call this model an AH-graph (Annotated, Hierarchical Graph). Multiple abstraction, on the other hand, consists of interwaving several AH-graphs together, enabling the possibility of abstracting in different ways the same ground information. Our multiple abstraction model is called a Multi-AH-graph and it has shown even higher computational efficiency in some situations [2],[3],[4].

We also have developed a heuristic optimization framework concerning points 1) and 2) [7],[2]. It is based on an evolutionary algorithm that produces at each time the best model of the environment, given the robot's operating experience. We have achieved several important results: first, the cost of constructing the model is spread along the working life of the robot, tending to be negligible over time (in contrast with more classical approaches that construct models through costly processes concentrated in a few moments); second, it acquires an important capacity of adaptation to environmental and robot operation variations (in contrast with approaches that pursue to optimize some static goal predefined by humans); and third, it is optimized not only with new acquired information, but also by taking the maximum advantage from past knowledge. Our framework obtains important improvements in the computational cost of planning tasks, even when the world or the tasks vary, and under any amplitude of those variations.

Finally, point 3) is the most recently problem we have dealt with in our research. We are using for that purpose techniques of anchoring [8], which maintain dynamic links between real objects or features of space and ground symbols of the model.

References

1. Brooks, R. A. A Robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation, **2** No. 1, March 1986, pp. 14-23; also MIT AI Memo 864, September 1985.
2. Fernandez-Madrigal J.A. and Gonzalez J. Multihierarchical Representation of Large-Scale Space. Applications to Mobile Robots, Series on Microprocessor-Based and Intelligent Systems Engineering **24**, Kluwer Academic Publishers. (2001)
3. Fernandez-Madrigal J.A. and Gonzalez J. Multihierarchical Graph Search. IEEE Transactions on Pattern Analysis and Machine Intelligence, **24**, no. 1, (2002).
4. Fernandez-Madrigal J.A., Galindo C., Gonzalez J. Assistive Navigation of a Robotic Wheelchair using a Multihierarchical Model of the Environment. Integrated Computer-Aided Engineering, **11** No. 4, (2004) pp. 309-322.
5. Galindo C., Fernandez-Madrigal J.A., Gonzalez J. Improving Efficiency in Mobile Robot Task Planning Through World Abstraction. IEEE Transactions on Robotics, **20** No. 4, (2004) pp. 677-690.
6. Fernandez-Madrigal J.A., Gonzalez J. Hierarchical Graph Search for Mobile Robot Path Planning. IEEE International Conference on Robotics and Automation (ICRA'98), Leuven, Belgium, (1998)
7. Galindo C., Fernandez-Madrigal J.A., Gonzalez J. Self-Adaptation of the Symbolic World Model of a Mobile Robot. An Evolution-Based Approach. 6th Int. FLINS Conf. on Applied Computational Intelligence, Blankenberg, Belgium, (2004).
8. Coradeschi S. and Saffiotti A. An introduction to the anchoring problem. Robotics and Autonomous Systems, **43** pp. 85-96, (2003).

Sequential Decision Making Under Uncertainty

Masoumeh Tabaeh Izadi

Reasoning and Learning Laboratory,
McGill University, Montreal, QC H3A 2A7 Canada
mtabae@cs.mcgill.ca

My main research centers around sequential decision making under uncertainty. In a complex dynamical system useful abstractions of knowledge can be essential to an autonomous agent for efficient decision making. Predictive State Representation, PSR, has been developed to provide a maintainable, self-verifiable and learnable representation of the knowledge of the world. I was very much intrigued by the PSR work, and started working on incorporating PSRs into POMDP control algorithms. Since the representational power of PSRs is equivalent to the belief state representation in POMDPs, one can imagine PSR planning algorithms, working in the context of controlling dynamical systems. In prior work [1] I developed an exact planning algorithm based on known PSR parameters. However, like all other exact algorithms, this approach has exponential complexity in the worst case. In preliminary experiments on a variety of standard domains, the empirical performance seems similar to belief-based planning.

I have also investigated ways of taking advantage of the PSR structure to improve the efficiency of the previously existed exact and approximation algorithms for POMDPs. My recent work on using core belief state in PBVI, Point-Based Value Iteration, [4] shows promising results in terms of value improvement[2]. Using the belief points lying in the intrinsic dimensions of reachable belief simplex allows us to abstract away redundancy in the problem definition where possible. Moreover the basis for the reachable simplex guides the beleif point selection of the algorithm to cover the reachable points rapidly.

Working along PSR research, I am currently studying structure exploration in problem domains using mathematical properties of predictive representation. PSR groups together states which behave similarly and it holds the promise of a more compact representation than POMDPs. We point out special cases in which strict reduction in the number of states is obtained by linear PSRs [3].

Another aspect of PSR study is learning the structure of the model. The learning algorithms developed for PSRs so far are based on a suitable choice of core tests. However, knowing the model dimensions (i.e. the core tests) is a big assumption and not always possible. Finding the core tests incrementally from interactions with an unknown system seems to be computationally as difficult as POMDP model leaning in general. One interesting line of thought could be considering extensions of tests from sufficiently large number of given histories and discovering the maximum possible core tests .

In summary, the objective of my research is using probabilistic approaches to assist intelligent agents in knowledge representation, reasoning and decision

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 360–361, 2005.

making, specifically to create learning and planning algorithms for predictive state representations.

References

1. Masoumeh T. Izadi, Doina Precup. A planning algorithm for Predicetieve State Representation. Proceedings of the 18th International Joint Conference on Artificial Intelligence. IJCAI 2003
2. Masoumeh T. Izadi, Ajit V. Rajwade, and Doina Precup. Using core beliefs for point-based value iteration. To appear in the proceedings of the 19th International Joint Conference on Artificial Intelligence. IJCAI 2005
3. Masoumeh T. Izadi, Doina Precup. Model reduction by linear PSRs . To appear in the proceedings of the 19th International Joint Conference on Artificial Intelligence. IJCAI 2005
4. Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithms for POMDPs. Proceedings of the 18th International Joint Conference on Artificial Intelligence. IJCAI 2003.

Automatic State Abstraction for Pathfinding in Real-Time Video Games

Nathan Sturtevant, Vadim Bulitko, and Michael Buro

Department of Computing Science, University of Alberta,
Edmonton, Alberta, T6G 2E8, Canada
{nathanst, bulitko, mburo}@cs.ualberta.ca

Abstract. Real-time video games are a unique domain for pathfinding and search. Traditional approaches to search have usually assumed static worlds with a single agent. But, in real-time video games there are multiple cooperative and adversarial agents. While the search space in most games is relatively small, algorithms are expected to plan in mere milliseconds. Thus, techniques such as abstraction are needed to effectively reason and act in these worlds. We provide an overview of the research we have completed in this area, as well as areas of current and future work.

1 Introduction

The worlds of popular real-time video games, such as Warcraft III or Command and Conquer are much richer than many traditional domains used as test-beds for Artificial Intelligence research. The most basic task in these games is to get one or more units from their current location to some destination. These games are not designed to showcase algorithms from AI, but to be interesting for people to play. This means that standard approaches from AI and search are usually too slow or memory-intensive to be effective in practice. We believe that abstraction is the key technique for meeting the real-time and other constraints posed by these domains.

2 Previous Work

We first provide an overview of completed work on applying abstraction to search. First, we have developed a simple method for automatically building abstractions from an underlying octile map. For the purposes of this discussion, we consider any map to be a graph, where a node is a tile in the underlying map, and an edge means an agent can pass directly between two tiles without going through another tile first.

Given a graph representation of a map, we build an abstract version of the map by reducing connected components into abstract nodes. Instead of using the approach of [1] where nodes are abstracted along with their neighbors, we only abstract groups of nodes that are fully connected cliques. In practice, this means that at most four nodes are abstracted in any one step. When building an abstraction offline, we process the space in a uniform manner so that the abstract space more closely represents

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 362–364, 2005.

the space it abstracts. This process can also be applied in an online fashion by assuming unknown portions of the map are empty and applying local repair as an agent explores the map.

Given an abstract representation of a search space, there are many methods that leverage this abstraction for quick pathfinding. The approach of Botea et. al. [2] is to abstract large sectors in the original space. When doing refinement, local smoothing is applied to account for error introduced by large sector sizes. Smoothing, however, is only applied to complete paths. Because we are interested in dynamic worlds, we do not always want to compute complete paths. Instead, we build partial paths at each level of abstraction refining them locally as needed. This allows us to spread the computational cost of path following evenly across path execution. So, if we are interrupted, we reduce lost computation efforts. These methods are described in detail in [3].

Abstraction necessarily introduces error, so we would like to learn about errors and correct them over repeated pathfinding experiences. Thus, we have taken the abstract search that is applied at each level of our abstraction and replaced the A* search with a learning search [4]. This allows us to learn much better heuristics in abstract space.

3 Current and Future Work

We are currently working towards four goals with regard to abstraction and search. First, a variety of ideas have been suggested for building abstractions of search spaces, including reductions based on cliques, local neighborhoods, large sectors, and triangulation. We are attempting to incorporate a more flexible abstraction module into our simulation framework so that we can more precisely quantify the advantages and trade-offs of different methods.

Second, we would like to generalize existing work to a single algorithm that can parameterize methods for search, so we can better evaluate which parameters for search work best on which problems and abstraction methods.

Third, general work on learning better heuristics is computationally expensive, because we must keep a large table of heuristic information between every pair of nodes in our search space. Storing heuristics in abstract space reduces this cost somewhat, but we are also looking into ways to reduce this cost further by selectively storing learned heuristic information at each level of abstraction. To this end we are actively developing high-performance learning methods for real-time heuristic search [5].

Finally, we are building stochastic and dynamic environments in which there are multiple cooperative and competitive units that must interact, so that we can measure how existing techniques scale to environments typical of real-time video games.

References

1. Holte, R., Perez, M., Zimmer, R., MacDonald, A.: Hierarchical A*: Searching abstraction hierarchies efficiently. In: AAAI/IAAI Vol. 1. (1996) 530–535
2. Botea, A., Müller, M., Schaeffer, J.: Near optimal hierarchical path–finding. Journal of Game Development **1(1)** (2004) 7–28

3. Sturtevant, N., Buro, M.: Partial pathfinding using map abstraction and refinement. In: Under Review. (2005)
4. Bulitko, V., Sturtevant, N., Kazakevich, M.: Speeding up learning in real-time search via state abstraction. In: Under Review. (2005)
5. Bulitko, V., Lee, G.: Learning in real-time heuristic search: A unifying framework. Under Review (2005)

Model-Based Search

Wheeler Ruml

Palo Alto Research Center, 3333 Coyote Hill Road,
Palo Alto, CA 94304 USA
ruml@parc.com

Abstract. I am interested in heuristic search and optimization, particularly algorithms that explicitly construct a model of the search space and attempt rational action with respect to this abstract model.

My main research interest is in heuristic search and optimization. In such problems, one explores a search tree (or graph) using various sources of information to find a good solution quickly. I am particularly interested in the setting when insufficient time is available to find and prove a solution optimal. I try to approach this problem from the perspective of rational action on the basis of statistical evidence. As a search algorithm expands nodes and consults its information sources, it can be seen as an agent exploring an unknown environment and gathering information on which to base further action. Therefore, the design of search algorithms can take inspiration from machine learning and decision theory.

I have designed four algorithms which could be said to exemplify these concerns to varying degrees. Two apply to any bounded-depth tree, requiring only a measure of solution quality at each leaf, one applies only to finite-domain constraint satisfaction problems, and the last applies to shortest-path problems.

Adaptive Probing [1]. This is an incomplete algorithm for combinatorial optimization problems. It conducts repeated probes into a search tree, always starting at the root and descending until it finds a leaf. The algorithm builds a model during search of the expected effect on the final solution cost of selecting each choice at each level of the tree ($d \times b$ parameters for a b-ary tree of depth d). At a decision point, the algorithm selects each choice with the probability that it leads to lower cost solutions. The sampling of the algorithm thus adapts over time according to the observed leaf costs.

Best-Leaf-First Search (BLFS) [2]. This is a complete version of adaptive problem. Again, a model of the expected cost of each choice is learned from observed data such as the solution costs at the leaves. The algorithm uses the model to visit leaves in an efficient approximation of increasing predicted cost. This generalizes depth-first search and the discrepancy search algorithms [3] and allows the use of search orders that would be difficult to program by hand. Experiments with several types of models are available in [4].

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 365–366, 2005.

Complete Local Search (CLS) [5]. This is a complete algorithm for CSPs, and SAT in particular. It uses learning to avoid local minima during search. At each local minimum, a new clause is learned which has the effect of smoothing the search space. We prove that the learning converges and no local minima will be left. Interestingly, the search is entirely based on following the local gradient, yet it is guaranteed to be complete. The algorithm always responds immediately to the learned information, yet this eventually results in the globally optimal solution. Unfortunately, the learning component can consume exponential space in the worst case. However, the algorithm shows good performance on large benchmark problems in practice.

Bugsy [6]. This algorithm is a variant of best-first search which explicitly balances solution cost and search time according to the user's specified utility function. Rather than requiring the user to separately train a termination policy, as one must when using an anytime algorithm, Bugsy requires no training data and terminates when it judges that further search is not worthwhile. This rational behavior depends on a learning component which transforms a given heuristic lower-bound function into an estimator of the cost-to-go.

References

1. Ruml, W.: Incomplete tree search using adaptive probing. In: Proceedings of IJCAI-01. (2001) 235–241
2. Ruml, W.: Heuristic search in bounded-depth trees: Best-leaf-first search. In: Working Notes of the AAAI-02 Workshop on Probabilistic Approaches in Search. (2002)
3. Korf, R.E.: Improved limited discrepancy search. In: Proceedings of AAAI-96, MIT Press (1996) 286–291
4. Ruml, W.: Adaptive Tree Search. PhD thesis, Harvard University (2002)
5. Fang, H., Ruml, W.: Complete local search for propositional satisfiability. In: Proceedings of AAAI-04. (2004)
6. Ruml, W., Crawford, E.: Best-first search when time matters. In: Working Notes of the IJCAI-05 Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains. (2005)

Learning Skills in Reinforcement Learning Using Relative Novelty

Özgür Şimşek and Andrew G. Barto

Department of Computer Science, University of Massachusetts,
Amherst, MA 01003-9264, USA
{ozgur, barto}@umass.cs.edu

Abstract. We present a method for automatically creating a set of useful temporally-extended actions, or *skills*, in reinforcement learning. Our method identifies states that allow the agent to transition to a different region of the state space—for example, a doorway between two rooms—and generates temporally-extended actions that efficiently take the agent to these states. In identifying such states we use the concept of *relative novelty*, a measure of how much short-term novelty a state introduces to the agent. The resulting algorithm is simple, has low computational complexity, and is shown to improve performance in a number of problems.

1 Introduction

Recent methods in Reinforcement Learning (RL) allow an agent to plan, act, and learn with high-level actions, or *skills*, that are closed-loop policies over lower-level actions [13, 16, 15, 2]. An example of a skill that people use is driving. People usually choose among a small set of high-level skills. For example, once people learn how to drive, they typically do not think in terms of the lower-level behaviors that are involved in driving. They simply choose between, for instance, driving and walking to work. This simplifies their lives dramatically. Furthermore, people continuously learn new skills and face each new problem armed with a set of skills they have learned in the past.

As it is with humans, the ability to automatically develop useful skills is an invaluable asset in an autonomous agent. Agents with such a capability have the potential to solve difficult problems that currently represent a challenge to the field of artificial intelligence. More importantly, such agents would be able to develop what White [18] has called *competency* over their environment—achieving a mastery over their domain that allows them to efficiently solve new problems as they arise using the knowledge and skills they acquired in the past [1].

A number of methods have been proposed for autonomous development of skills in RL. One approach is to search for commonly occurring subpolicies in solutions to a set of tasks and to define skills with corresponding policies [17, 14]. A second approach is to identify subgoals—states that are useful to reach—and generate skills that take the agent efficiently to these subgoals. Subgoals proposed

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 367–374, 2005.

in the literature include states that are visited frequently or that have a high reward gradient [3], states that are visited frequently on successful trajectories but not on unsuccessful ones [9], and states that lie between densely-connected regions of the state space [10, 8, 12]. In addition, Hengst [5] has used the notion of a subgoal in performing temporal and spatial abstraction simultaneously, defining subgoals to be those states that lead to transitions that the agent can not correctly represent or predict at the current level of state abstraction. In this paper, we describe the *relative novelty algorithm* (RN), a subgoal-based method for creating a set of useful skills. We provide only a brief overview here; a more detailed description can be found in Şimşek & Barto [11].

The subgoals RN seeks to identify are states that allow the agent to transition to a part of the state space that is otherwise unavailable or difficult to reach from its current region. They are called *access states* and are closely related to the subgoals in [9, 10, 8, 12]. A simple navigational example is a doorway between two rooms. Easy access to these states allows more efficient exploration of the state space by providing direct access to those regions that are difficult to reach. Furthermore, access states are useful not only in solving a single, isolated task, but also in solving a variety of problems in the same domain—getting to the doorway is useful regardless of what the agent needs to do in the other room.

While navigational domains are rich sources of access states—and are useful in conveying their intuitive appeal—access states capture a certain type of connectivity structure of the state-transition graph that exists in a broader class of problems. For example, completion of a subtask in a sequential task is an access state; so is building a tool that makes possible a new set of activities for the agent.

2 The Relative Novelty Algorithm

RN is based on our intuition that access states will be more likely than other states to introduce short-term novelty, i.e., to mediate a transition to a region that the agent has not visited recently. Below we first define relative novelty and explain how RN identifies subgoals and generates skills that take the agent efficiently to these subgoals. Throughout our discussion, we refer to access states as *targets*, and to other states as *non-targets*.

2.1 Relative Novelty

We relate novelty to how frequently a state is visited since a designated start time. We define the novelty of a discrete state s to be equal to $\frac{1}{\sqrt{n_s}}$, where n_s is the number of times it has been visited. The novelty of a set S of states is $\frac{1}{\sqrt{\bar{n}_S}}$, where $\bar{n}_S$ is the mean number of times states in S have been visited. With this definition, the novelty of a state equals 1 when it is first visited, decays with each succeeding visit, and approaches 0 in the limit. Visitation frequencies are reset periodically because we are interested in a short-term measure of novelty.

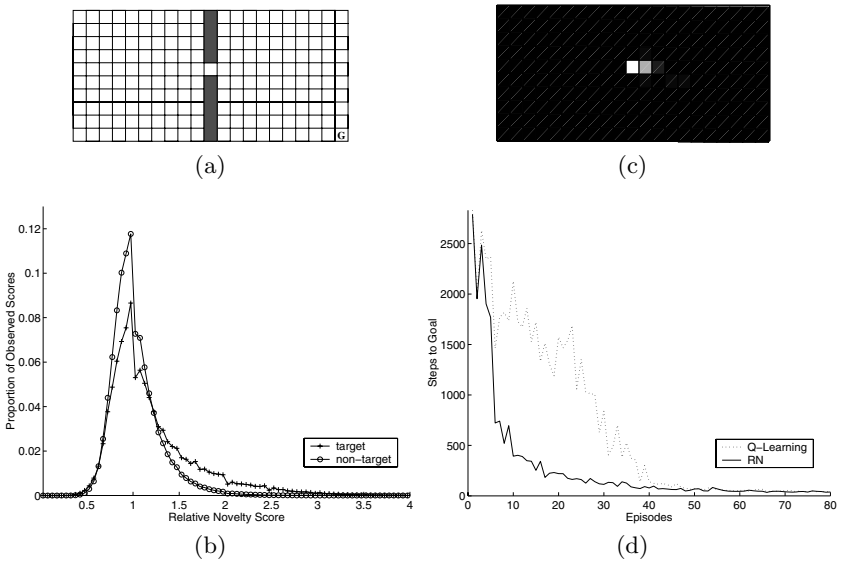


Fig. 1. (a) Two-Room gridworld environment, (b) Distribution of relative novelty scores for target and non-target states, (c) Subgoals identified, (d) Mean steps to goal

We define the relative novelty of a state in a transition sequence to be the ratio of the novelty of states that followed it (including itself) to the novelty of the states that preceded it. The number of forward and backward transitions to take into account in computing this score is a parameter called the *novelty lag* (l_n). A state will typically have a different relative novelty score each time it is visited.

Our intuition suggests that the distribution of relative novelty scores of targets will be different than that of non-targets. More specifically, we expect targets to have higher scores more frequently. We tested this hypothesis in a simple domain, the two-room gridworld shown in figure 1a. The actions were the usual `north`, `south`, `east`, `west`. We ignored the goal state and had the agent perform a 1000-step random walk 1000 times, starting each at a random grid location.

This domain has a single target state that fits our definition of a subgoal—the doorway between the two rooms. Figure 1b shows the distribution of relative novelty scores for the doorway and for other states, using a novelty lag of 7. The figure reveals that the distributions are indeed different for this domain. Both distributions peak around a relative novelty score of 1, indicating approximately equal novelty scores preceding and following a state, but the target distribution has a heavier tail. Repeated experiments with different novelty lags and room sizes showed a similar discrepancy in relative novelty scores of the doorway and other states. This discrepancy is the basis of our subgoal discovery method, which we discuss in the next section.

2.2 Subgoal Discovery

We formulate the subgoal discovery task for an on-line RL agent as a classification problem. With each transition, the agent observes a new relative novelty score for some state s and wishes to classify s as target (T) or non-target (N), based on not only the current score, but *all* scores observed so far for s. If class-conditional relative novelty distributions are known, this classification task is straightforward using Bayesian decision theory [4]. Assigning an appropriate cost to two possible types of error—classifying a target as non-target (miss) or a non-target as target (false alarm)—and minimizing total cost gives rise to the following decision rule:

Label state as target if

$$\frac{P\{(s_1,..,s_n)|T\}}{P\{(s_1,..,s_n)|N\}} > \frac{\lambda_{fa}}{\lambda_{miss}} \cdot \frac{P\{N\}}{P\{T\}} \tag{1}$$

where $(s_1,..,s_n)$ are the relative novelty scores observed for the state, $P\{i\}$ is the prior probability of a state of type i, λ_{fa} is the cost assigned to a false alarm, and λ_{miss} is the cost assigned to a miss.

We further simplify this rule by converting the continuous relative novelty score to a binary feature x, where x equals 1 if score is greater than a threshold (which we call the *relative novelty threshold* and denote by t_{RN}) and 0 otherwise. This is motivated by our observation that the distributions differ mainly in their tail, which suggests that the appropriate choice of a threshold would capture enough information to construct a good classifier. Assuming independent observations of relative novelty, we obtain the following decision rule:

Label state as target if

$$\frac{n_1}{n} > \frac{\ln\frac{1-q}{1-p}}{\ln\frac{p(1-q)}{q(1-p)}} + \frac{1}{n}\frac{\ln(\frac{\lambda_{fa}}{\lambda_{miss}} \cdot \frac{p(N)}{p(T)})}{\ln\frac{p(1-q)}{q(1-p)}} \tag{2}$$

where $p = P\{x = 1|T\}$, $q = P\{x = 1|N\}$, n_1 is the number of observations with $x = 1$, and n is the total number of observations.

The procedure for identifying subgoals is as follows:

1. (Off-line). Estimate the class-conditional relative novelty distributions using agent's experience, if possible, in a small part of the actual task, and otherwise in a corpus of environments. Determine the value of t_{RN} using a receiver operating characteristic (ROC) curve analysis [4]. Compute p, q given the class-conditional relative novelty distributions and t_{RN}. Determine $\frac{\lambda_{fa}}{\lambda_{miss}} \cdot \frac{P\{N\}}{P\{T\}}$.
2. (On-line). Evaluate decision rule 2 periodically, possibly with each new state transition, considering *all* of the relative novelty scores observed for a given state.

The subgoal discovery method of RN has a time complexity of $O(1)$—RN examines only the most recent experiences of the agent, so the computational cost does not grow with the number of states.

2.3 Generating Skills

We represent skills using the options framework [15, 16]. When a new subgoal is identified, RN generates an option whose policy efficiently takes the agent to this subgoal. The option's initiation set consists of those states that were visited shortly before the subgoal state registered a relative novelty score higher than t_{RN}. The number of past transitions to include in this set is determined by a parameter, the *option lag* (l_o). The option's policy is specified through an RL process employing action replay [7] using a pseudo reward function [2]. The policy learned takes the agent to the subgoal state in as few time steps as possible while remaining in the option's initiation set. The option terminates with probability 1 if the agent reaches the subgoal, or if the agent leaves the initiation set; otherwise, it terminates with probability 0.

3 Experimental Results

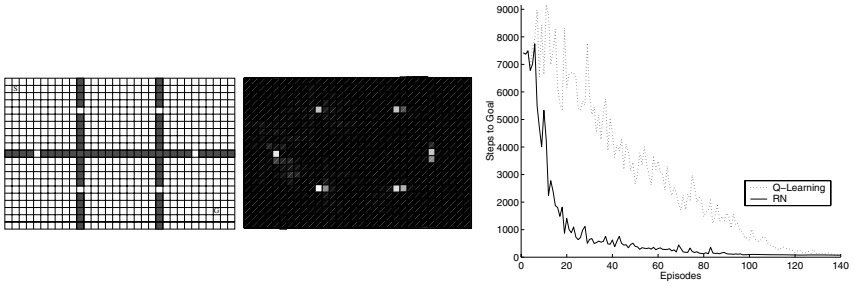
Below we present experimental results in a number of domains. The off-line part of the algorithm was conducted only once, using the data obtained from the random walk in the two-room gridworld discussed earlier in Section 2.1. Using an ROC curve analysis, t_{RN} was set to 2, which lead to a p value of 0.0712 and a q value of 0.0056. Other parameter settings were as follows: $\frac{\lambda_{fa}}{\lambda_{miss}} = 100$, $\frac{p(N)}{p(T)} = 100$, $l_n = 7$, $l_o = 10$. No limit was set on the number of options that could be generated; and no filter was employed to exclude certain states from being identified as subgoals. In all of our experiments, the agent used Q-learning with ϵ-greedy exploration with $\epsilon = 0.1$. The learning rate (α) was kept constant at 0.05; initial Q-values were 0.

3.1 Rooms

Our first example is the two-room gridworld in figure 1a, the domain that was used to estimate the class-conditional relative novelty distributions. Performance on this task shows how well the algorithm can do given almost perfect estimates of the class-conditional relative novelty distributions.

The agent started each episode on a random square in the west room; the goal was the grid square on the Southeast corner of the grid. The four primitive actions—`north`, `south`, `east`, `west`—moved the agent in the intended direction with probability 0.9, and in a uniform random direction with probability 0.1. If the direction of movement was blocked, the agent remained in the same location. The agent received a reward of 1 at the goal state, and a reward of 0 at all other states. The discount factor was 0.9.

Figure 1c shows a visual representation of the location and frequency of the subgoals identified in 30 runs. The color of a square in this figure corresponds to the number of times it was identified as a subgoal, with lighter colors indicating larger numbers. The state that was identified as a subgoal most frequently was the doorway—in 25 of the 30 runs. Mean number of subgoals identified per run was 1.6; 96% of the subgoals were within two steps of the doorway. Figure 1d



(a) (b) (c)

Fig. 2. (a) Six-Room gridworld, (b) Subgoals identified, (c) Mean steps to goal

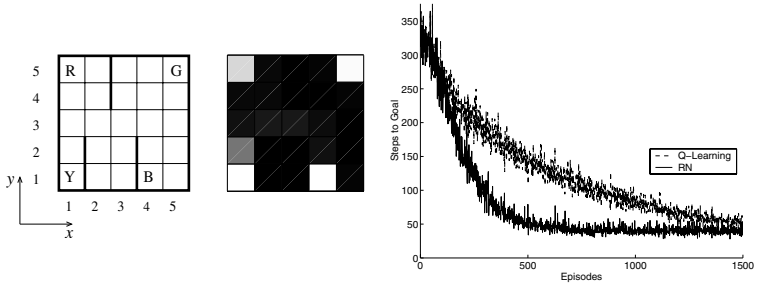
shows the mean number of steps taken to reach the goal state, with and without RN. The figure reveals that RN identified useful subgoals and showed a marked improvement in performance within 10 episodes.

Our second example is the larger gridworld environment shown in Figure 2a. Figures 2b and 2c show the results of 30 runs. Mean number of subgoals identified per run was 15.3; of the subgoals identified, 30% were target states and 24% were states one transition away from the targets. As in the previous example, the options generated drastically improved the agent's performance.

3.2 Taxi Task

Our next example is the taxi domain [2] depicted in Figure 3a. The task is to pick-up and deliver a passenger to her destination on a 5×5 grid. The source and destination are randomly and independently chosen in each episode, from among the set of squares marked by R, G, B, Y. The primitive actions are `north`, `east`, `south`, `west`, `pick-up`, `put-down`. The navigation actions succeed in moving the taxi in the intended direction with probability 0.80; with probability 0.20, the action takes the taxi to the right or left of the intended direction. The action `pick-up` places the passenger in the taxi if the taxi is at the same grid location as the passenger; otherwise it has no effect. Similarly, `put-down` delivers the passenger if the passenger is inside the taxi and the taxi is at the destination; otherwise it has no effect. Reward is -1 for each action, an additional $+20$ for passenger delivery, and an additional -10 for an unsuccessful `pick-up` or `put-down` action.

We evaluated the performance of RN in 100 runs. Figure 3b shows a visual representation of the grid location of the subgoals, ignoring the other two state variables. Mean number of subgoals identified per run was 25.8. Of these, 78% were subgoals that correspond to getting to the passenger location and picking up the passenger. Another 14% were states that are one transition away from these. Navigational bottlenecks—grid squares (2,3) and (3,3)—accounted for 4% of the subgoals. Altogether, these add up to 96% of the subgoals identified. Mean number of steps to complete the task is shown in Figure 3c.



(a) (b) (c)

Fig. 3. (a) The taxi domain, (b) Subgoals identified (showing only the grid location variable), (c) Mean steps to goal

4 Discussion

The algorithm introduced here is a simple implementation of the general idea of using a measure of short-term novelty to identify states that may form useful target states for a collection of skills. The intuition is that if the ease of reaching such states is improved, the agent's access to unexplored regions of the state space will improve, thus leading to more efficient exploration. A key aspect of the algorithm is that the process of identifying subgoals is not dependent on the reward function of the overall task. Indeed, there may be no such reward function. This implies that the method can facilitate transfer among multiple tasks with disparate reward functions and that it can provide potentially useful abstract actions before any particular task has been solved (in cases where "solving a task" has a well-defined meaning). This property is essential if an automatic abstraction method is to be useful in extending the utility of RL to complex real-world tasks.

Finally, we comment that various concepts of novelty are closely linked to motivation and reward in animals (e.g., Kakade & Dayan, 2001). The use of novelty measures to drive the automatic creation of hierarchical behavior architectures may provide useful computational interpretations of novelty-related animal behavior.

Acknowledgments

This work was supported by the National Science Foundation under Grant No.CCF-0432143 and by a subcontract from Rutgers University, Computer Science Department, under Award number HR0011-04-1-0050 from DARPA. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

1. A. G. Barto, S. Singh, and N. Chentanez. Intrinsically motivated learning of hierarchical collections of skills. In *Proceedings of the Third International Conference on Developmental Learning*, 2004.
2. T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.
3. B. Digney. Learning hierarchical control structure for multiple tasks and changing environments. In *From Animals to Animats 5: The Fifth Conference on the Simulation of Adaptive Behaviour*. The MIT Press, 1998.
4. R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley, New York, 2001.
5. B. Hengst. Discovering hierarchy in reinforcement learning with HEXQ. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 243–250. Morgan Kaufmann, 2002.
6. S. Kakade and P. Dayan. Dopamine bonuses. In *Advances in Neural Information Processing Systems*, volume 13, pages 131–137. MIT Press, 2001.
7. L. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, 1992.
8. S. Mannor, I. Menache, A. Hoze, and U. Klein. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the Twenty-First International Conference on Machine Learning*, 2004.
9. A. McGovern and A. G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 361–368. Morgan Kaufmann, 2001.
10. I. Menache, S. Mannor, and N. Shimkin. Q-Cut - Dynamic discovery of sub-goals in reinforcement learning. In *Proceedings of the Thirteenth European Conference on Machine Learning*, volume 2430 of *Lecture Notes in Computer Science*, pages 295–306. Springer, 2002.
11. Özgür Şimşek and A. G. Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 751–758. ACM Press, 2004.
12. Özgür Şimşek, A. P. Wolfe, and A. G. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the Twenty-Second International Conference on Machine Learning*, To appear.
13. B. R. Parr. *Hierarchical Control and Learning for Markov Decision Processes*. PhD thesis, Computer Science Division, University of California, Berkeley, 1998.
14. M. Pickett and A. G. Barto. PolicyBlocks: An algorithm for creating useful macro-actions in reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 506–513. Morgan Kaufmann, 2002.
15. D. Precup. *Temporal abstraction in reinforcement learning*. PhD thesis, University of Massachusetts Amherst, 2000.
16. R. S. Sutton, D. Precup, and S. P. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.
17. S. Thrun and A. Schwartz. Finding structure in reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 7, pages 385–392. MIT Press, 1995.
18. R. W. White. Motivation reconsidered: The concept of competence. *Psychological Review*, 66:297–333, 1959.

Author Index