Lecture Notes in Artificial Intelligence 2371

Subseries of Lecture Notes in Computer Science
Edited by J. G. Carbonell and J. Siekmann

Lecture Notes in Computer Science
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
*Berlin*
*Heidelberg*
*New York*
*Barcelona*
*Hong Kong*
*London*
*Milan*
*Paris*
*Tokyo*

Sven Koenig  Robert C. Holte (Eds.)

# Abstraction, Reformulation, and Approximation

5th International Symposium, SARA 2002
Kananaskis, Alberta, Canada August 2-4, 2002
Proceedings

Springer

Series Editors

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

Volume Editors

Sven Koenig
College of Computing, Georgia Institute of Technology
801 Atlantic Dr NW, Atlanta, GA 30332-0280, USA
E-mail: skoenig@cc.gatech.edu

Robert C. Holte
Universtity of Alberta, Department of Computing Science
2-21 Athabasca Hall, Edmonton, Alberta
T6G 2E8, Canada
E-mail: holte@cs.ualberta.ca

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Abstraction, reformulation, and approximation : 5th international symposium ; proceedings / SARA 2002, Kananaskis, Alberta, Canada, August 2 - 4, 2002. Sven Koenig ; Robert C. Holte (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Tokyo : Springer, 2002
(Lecture notes in computer science ; Vol. 2371 : Lecture notes in artificial intelligence)
ISBN 3-540-43941-2

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

ISSN 0302-9743
ISBN 3-540-43941-2 Springer-Verlag Berlin Heidelberg New York



Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de


Printed in Germany

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Stefan Sossna e.K.
Printed on acid-free paper SPIN: 10870376 06/3142 5 4 3 2 1 0

# Preface

It has been recognized since the inception of Artificial Intelligence (AI) that abstractions, problem reformulations, and approximations (AR&A) are central to human common-sense reasoning and problem solving and to the ability of systems to reason effectively in complex domains. AR&A techniques have been used to solve a variety of tasks, including automatic programming, constraint satisfaction, design, diagnosis, machine learning, search, planning, reasoning, game playing, scheduling, and theorem proving. The primary purpose of AR&A techniques in such settings is to overcome computational intractability. In addition, AR&A techniques are useful for accelerating learning and for summarizing sets of solutions.

This volume contains the proceedings of SARA 2002, the fifth Symposium on Abstraction, Reformulation, and Approximation, held at Kananaskis Mountain Lodge, Kananaskis Village, Alberta (Canada), August 2-4, 2002. The SARA series is the continuation of two separate threads of workshops: AAAI workshops in 1990 and 1992, and an ad hoc series beginning with the "Knowledge Compilation" workshop in 1986 and the "Change of Representation and Inductive Bias" workshop in 1988 with followup workshops in 1990 and 1992. The two workshop series merged in 1994 to form the first SARA. Subsequent SARAs were held in 1995, 1998, and 2000.

SARA's aim is to provide a forum for intensive interaction among researchers in all areas of AI with an interest in the different aspects of AR&A techniques. The diverse backgrounds of participants leads to a rich and lively exchange of ideas, and a transfer of techniques and experience between researchers who might otherwise not be aware of each other's work.

SARA has a tradition of inviting distinguished researchers from diverse areas to give technical keynote talks of a survey nature. SARA 2002 has two keynote speakers from established SARA areas: Sridhar Mahadevan will speak about abstraction and reinforcement learning and Derek Long about reformulation in planning. SARA 2002 also has two keynote speakers from areas that have not been strongly represented at previous SARAs: Bob Kurshan will survey the use of abstraction in model-checking and Aristide Mingozzi will survey state space relaxation and search strategies in dynamic programming.

The papers in this volume are representative of the range of AR&A techniques and their applications. We would like to thank the authors and the keynote speakers for their efforts in preparing high quality technical papers and presentations accessible to a general audience, and thank the program committee and anonymous reviewers for the time and effort they invested to provide constructive feedback to the authors. We are very grateful for the assistance we received in organizing SARA 2002 from Susan Jackson, Sunrose Ko, Yngvi Bjornsson, Rob Lake, and Shirley Mitchell. Judith Chomitz and Tania Seib at the Kananaskis Mountain Lodge were a pleasure to work with. We would like to express a special thanks to Berthe Choueiry for her advice, suggestions, and support.

Several organizations provided financial support or assistance which greatly enhanced the richness of the SARA experience, and for which all SARA 2002 participants owe thanks: the American Association for Artificial Intelligence (AAAI), NASA Ames Research Center, the Pacific Institute for the Mathematical Sciences (PIMS), the University of Alberta, and Georgia Institute of Technology. SARA 2002 is a AAAI affiliate.

July 2002

Sven Koenig
Robert C. Holte

# Organization

**Symposium Co-chairs**

Sven Koenig, Georgia Institute of Technology
Robert C. Holte, University of Alberta

**Program Committee**

Ralph Bergmann, University of Hildesheim
Yngvi Bjornsson, University of Alberta
Craig Boutilier, University of Toronto
Berthe Y. Choueiry, University of Nebraska-Lincoln
Stefan Edelkamp, Albert-Ludwigs-Universität Freiburg
Tom Ellman, Vassar College
Boi V. Faltings, Swiss Federal Institute of Technology in Lausanne
Jeremy Frank, NASA Ames
Eugene C. Freuder, Cork Constraint Computation Centre
Mike Genesereth, Stanford University
Lise Getoor, University of Maryland
Fausto Giunchiglia, University of Trento and ITC-IRST
Henry Kautz, University of Washington
Terran Lane, MIT
Michael Lowry, NASA Ames Research Center
Hiroshi Motoda, Osaka University
Pandurang Nayak, PurpleYogi.com
Doina Precup, McGill University
Peter Revesz, University of Nebraska-Lincoln
Lorenza Saitta, Università del Piemonte Orientale
Bart Selman, Cornell University
Barbara Smith, University of Huddersfield
Marco Valtorta, University of South Carolina at Columbia
Jeffrey Van Baalen, University of Wyoming
Toby Walsh, University of York
Weixiong Zhang, Washington University (St. Louis)
Robert Zimmer, Goldsmiths College, University of London
Jean-Daniel Zucker, Université Pierre & Marie Curie

**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, University of York

**Sponsoring Institutions**

The American Association of Artificial Intelligence (AAAI)
NASA Ames Research Center
The Pacific Institute for the Mathematical Sciences (PIMS)
The University of Alberta
Georgia Institute of Technology

# Table of Contents

## Invited Presentations



## Full Presentations

## Short Presentations

## Research Summaries

# Model Checking and Abstraction*

Robert P. Kurshan

Cadence Design Systems
rkurshan@cadence.com

How can a computer program developer ensure that a program actually implements its intended purpose? This article describes a method for checking the correctness of certain types of computer programs. The method is used commercially in the development of programs implemented as integrated circuits, and is applicable to the development of "control-intensive" software programs as well. "Divide-and-conquer" techniques central to this method apply to a broad range of program verification methodologies.

Classical methods for testing and quality control no longer are sufficient to protect us from communication network collapses, fatalities from medical machinery malfunction, rocket guidance failure, or a half-billion dollar commercial loss due to incorrect arithmetic in a popular integrated circuit. These sensational examples are only the headline cases. Behind them are multitudes of mundane programs whose failures merely infuriate their users and cause increased costs to their producers.

A source of such problems is the growth in program complexity. The more a program controls, the more types of interactions it supports. For example, the telephone "call-forwarding" service (forwarding incoming calls to a customer-designated number) interacts with the "billing" program that must determine whether the forwarding number or the calling number gets charged for the additional connection to the customer-designated number. At the same time, call-forwarding interacts with the "connection" program that deals with the issue of what to do in case the called number is busy, but the ultimate forward destination is free. One property to check is that a call is billed to the customer if and only if the connection is completed. If the call connection and billing programs interact incorrectly, a called number that was busy and then became free could appear busy to one program and free to the other, resulting in an unbilled service or an unwarranted charge, depending upon their order of execution.

If a program includes $n$ inter-related control functions with more than one state, the resulting program may need to support $2^n$ distinct combinations of interactions, any of which may harbor a potential unexpected peculiarity. When $n$ is very small, the developer can visualize all the combinations, and deal with them individually. Since the size of a program tends to be proportional to the number of functions it includes (one program block per function), the number of program interactions as a function of program size may grow exponentially. As a result, the developer can use only a very small proportion of the possible program

* This article is derived from my article entitled *Program Verification*, which appeared in the Notices of the American Mathematical Society, vol. 47, May 2000, and is excerpted here with their permission.

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 1–17, 2002.

interactions to guide the development and testing of the program. When some unconsidered combination produces an eccentric behavior, the result may be a "bug".

While a computer could be programmed to check a program under development for eccentric behavior by searching exhaustively through all combinations of program interactions, the exponential growth could soon cause the computer to exceed available time or memory. On account of the potential for interactions, adding a few functions to a program can substantially alter its testability and thus viability. From the program developer's perspective, this is unsatisfactory. If the program is developed carefully, however, the correct behavior of each of the $n$ individual control functions of the program can be checked in a way such that the complexity of the checks grow more or less proportionally to the size of the program.

## 1 Overview

This article presents some key ideas of "program verification". It focuses on the "reduction and decomposition" process, which addresses the problem of how to verify a program automatically in a way that the computational complexity scales tractably with increasing program size. It does not attempt to survey the field. A survey of automaton-based verification is included in [9]; for a survey of logic-based verification, see [4,7].

### 1.1 Reduction and Decomposition

"Reduction" and "decomposition" circumvent the exponential cost of checking correctness. Checking each control function separately, *reduction* refers to an algorithm through which the program to be checked is replaced, relative to the respective control function, with a simpler program that it is sufficient to check. The simpler program is always an abstraction of the original program. For example, to check the operation of call-forwarding, the part of the program that performs billing may be largely ignored, except to the extent that it may interfere with call-forwarding. The reduced (more abstract) program is checked through an algorithm that analyzes every possible program state. In order for a computer to implement this effectively, the reduced program must be sufficiently simple.

*Decomposition* refers to checking that a given control function is correctly implemented, by splitting the function into several simpler-to-check "subfunctions". Taken together, the subfunctions implement the original function. A subfunction is "simpler to check" if it gives rise to a simpler reduction than the original function. Decomposition and reduction thus are used together as divide-and-conquer techniques. As an example, call-forwarding may be decomposed into its "setup" where the customer designates the new destination for incoming calls, and "execution" where an incoming call gets forwarded to the designated destination. In verifying setup, the reduced program can in addition ignore most of the parts

of the original program that perform the execution subfunction. Likewise in the verification of the execution subfunction, most parts of the program that deal with setup can be ignored. A computer program can check that these subfunctions collectively implement the original function. In general, it is a manual step to decide how to decompose a function.

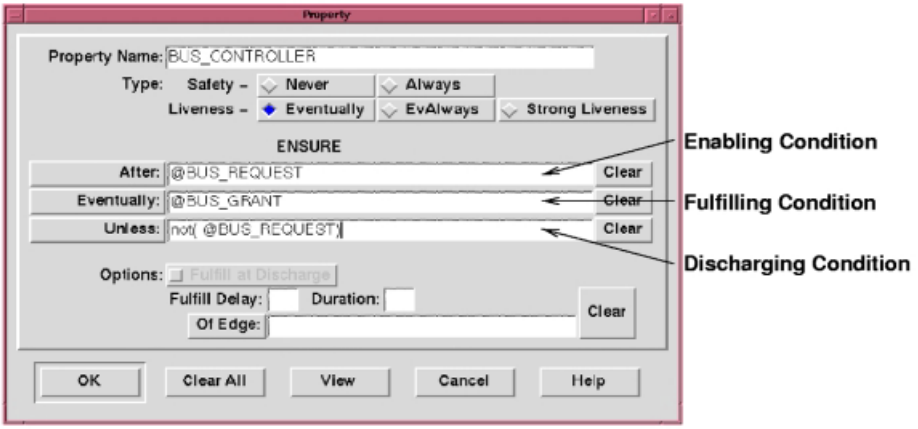



**Fig. 1.** In the commercial model-checking tool FormalCheck [11], required program operation is specified in terms of an *enabling condition* that determines the onset of a requirement specified by a *fulfilling condition* that must hold unless released by a *discharging condition, ad infinitum*. Each such specification is represented internally by an $\omega$-automaton. Any property expressible with an $\omega$-automaton can be expressed by a collection (logical conjunction) of such specifications.

## 1.2 Formal Verification

Program verification is also called "formal verification" to contrast it with conventional testing, which lacks a precise description of what was tested. Formal verification begins with a formal statement of some high-level purpose of the program, and determines whether that purpose is supported in every possible execution of the program. This is accomplished by analyzing the logic of the program, not by executing it. Since formal verification can account for every possible program execution, it is more reliable than conventional testing. There are numerous accounts of bugs that could have caused significant commercial damage had they remained, that were found quickly with formal verification after having been missed by conventional testing.

As a program's purpose evolves, the influence of one part on another grows, so correcting defects becomes increasingly costly. A common observation is that the cost of fixing a bug doubles with each successive stage of development. Finding

and correcting program errors early in the design cycle thus can decrease development time and cost significantly. Conventional testing requires the creation of program-specific test benches, so it tends to be deferred until late in the program development cycle, when the program development has stabilized. By contrast, formal verification algorithms are independent of the program to be verified. Therefore, formal verification can be applied early in the program-development cycle.

Within the last ten years research in formal verification has found its way to commercial products that implement program verification algorithms. For the present, these products are specialized mainly for industries that manufacture integrated circuits. However, the program verification methodologies described here can be applied to significant portions of general software development, and this is the direction of the industry.

### 1.3 Verification Methodologies

Formal verification refers not just to a single methodology but to a spectrum of methodologies. In this spectrum, there is a tradeoff between *expressiveness*—what program properties can be described and analyzed—and *degree of automation*—the extent to which the verification can be automated with an efficient algorithm.

The most expressive form of formal verification could be said to be mathematical argument. However, the reliability of mathematical argument is based upon peer review [5]. It is unlikely that too many peers could be mustered to check a 100-page proof of the correctness of an integrated circuit. Moreover, it is unlikely that the circuit manufacturer would be willing to wait for the years it might take to produce such a proof in the first place.

This latter problem is shared by automated theorem-proving [2]. The dream of writing a computer program capable of automatically proving marvelous theorems never was completely realized. Instead, the "automatic theorem-provers" in fact are proof checkers. While often quite sophisticated, powerful, and useful, automatic theorem-provers mainly require the user to come up with the proof, which must be entered into the "theorem-proving" program using the special notation accepted by the program. Theorem-provers thus do not presently support a methodology that can keep up broadly with a commercial development cycle.

This disappointment led researchers to investigate the other end of the spectrum: methodologies that sacrifice expressiveness in favor of automation. Among the first to discuss this strategy were E. M. Clarke and E. A. Emerson, who, in 1980, proposed a limited but completely algorithmic form of verification they called "model-checking" [3]. Their mode of verification was founded in a logic that supported a very simple model-satisfaction check. Around the same time and independently, J.-P. Quielle and J. Sifakis made a similar proposal.

In 1982, ignorant of the above work, I proposed a verification method based on automata. Eventually (ten years later!) the method was implemented in a

commercial tool (Figure 1) that is marketed to manufacturers of integrated circuits. Since for the moment at least this is the dominant commercial model-checking tool, I will exercise a certain prerogative and restrict my discussion of model-checking to automata. There is no loss of generality in using automata, as logic-based model-checking can be reduced to automaton-based model-checking (and *vice versa*).

### 1.4 Program State

In the von Neumann stored program model, "instructions" update "data" stored in "memory registers", in a sequential fashion. "Program variables" designate memory locations and their data contents, and also define via "assignment expressions" the rules by which the data stored in the variables get modified.

The *program state* is the vector of simultaneous values of all program variables. It captures the entire instantaneous condition of the program, independent of its history. A program's *behavior* is captured by its succession of states. However, the computer or integrated circuit that implements the program makes it effectively impossible (on purpose)—or renders it unnecessary—to discern the precise sequential evolution of certain "transient" states. The program model should reflect this, and moreover can exploit it to reduce the computational complexity of verification. For example, for variables `x`, `y`, and `z`, suppose `x` is assigned to take the value of `y + 1` after `y` is assigned to take the value of `z + 1`. An analysis of the program may reveal that with regard to its implementation, `x` effectively is or can be treated as a "macro" or synonym for `y + 1` and likewise, `y` for `z + 1`, simplifying the two successive assignments to a simultaneous assignment of the value of `z + 1` to `y` and of `z + 2` to `x`.

Deciding which assignments are to be modeled as simultaneous must be done with great care in order to preserve the semantics of the implemented program. Commercial model-checkers perform this task algorithmically. In the program model, a *sequential* variable is one that attains its value after (rather than simultaneously with) the value attained by its assignment expression. Typically, if a variable `x` is assigned the value of an expression that depends on `x`, for example, if `x` is assigned the value of `x + 1`, then `x` would be designated a sequential variable. The new value of `x` would then be modeled as succeeding the prior value of `x`.

Limiting the number of sequential variables in a program is very important for the performance of an integrated circuit, and for the performance of model-checking too. There is a multibillion dollar industry for "synthesis" tools that help create an integrated circuit layout automatically from its defining program, focused in important part on this partitioning problem.

The nonsequential program variables are called *combinational* variables (to use the hardware term). These hold the values of respective "transient" steps in a computation, such as the value of `y` used in `x` above. They include the *primary inputs*, which are variables whose values are assigned according to decisions made outside of the program, *e.g.*, which keyboard key a human pushes. Typically, program outputs also are represented by combinational variables.

In the program model, the program state is partitioned into the *sequential state* defined by the values of the sequential variables, and the *combinational state* defined by the remaining (combinational) variables. The combinational state vector $\mathbf{C}$ is given as a function $\mathbf{C} = \mathbf{f}(\mathbf{S})$ of the sequential state vector $\mathbf{S}$. The sequential state $\mathbf{S}$ is initialized and each successive value

$$\mathbf{S}' = \mathbf{F}(\mathbf{S}, \mathbf{C}) \tag{1}$$

is expressed in terms of its current value $\mathbf{S}$ and the combinational state.

### 1.5 Nondeterminism

In a model of the program, the assignment of primary inputs must be abstracted to account for all possible assignments. Abstraction is used more generally in program verification to simplify parts of a program whose detailed function is irrelevant to the verification objective. Abstracting duration in respective asynchronous parallel processes can result in a simpler model that is useful when the required behavior is independent of the relative execution speeds of the processes.

An effective way to abstract a program is through the use of *nondeterminism* in program variable assignments. If the function $\mathbf{f}(\mathbf{S})$ is allowed to be multivalued, at each sequential state $\mathbf{S}$ a combinational variable may assume several values in its range. This gives rise to a set of program behaviors or "lifetimes". Each nondeterministic assignment splits the execution of the program model into separate respective behaviors, *ad infinitum*.

Without nondeterminism, a program would have only one behavior, and its analysis would be simple. Although it is the nondeterminism that makes program analysis difficult, there is no real alternative with regard to the primary inputs. It also may be simpler and sufficient to model with a nondeterministic assignment certain variables that *are* assigned within the actual program. Imagine a program subroutine that performs a complex computation producing a nonzero result, writing the result to the program variable `v`. Suppose the program is required to perform some task unrelated to the computation each time the subroutine completes its calculation, the completion indicated by $\mathtt{v} \neq 0$. In order to verify the property $P$ that whenever $\mathtt{v} \neq 0$ the task is correctly performed, the computation that assigns `v` is irrelevant. To verify $P$, it is sufficient to have the program model assign `v` nondeterministically to 0 or 1, where $\mathtt{v} = 1$ is an abstraction that stands for all the nonzero values of the variable `v` in the original program. In this abstraction, the verification routine can determine, through a localization reduction described below, that the complex computation is irrelevant, and exclude it from the analysis leading to the verification of $P$. (That computation will be relevant to other properties, with respect to which other portions of the program may be irrelevant.) If $P$ is verified in the abstract model and the computation that assigns `v` in the original program is subsequently altered, it is unnecessary to reverify $P$.

If $\mathtt{w} \neq 0$ signals the conclusion of another program, then the order in which `v` and `w` assume nonzero values indicates the relative speeds of the associated

programs. Assigning `v` nondeterministically allows it to become nonzero both before and after `w` does. A property verified with this abstraction will hold for alternative implementations that vary the relative speeds of the programs.

Nondeterministic assignment may introduce additional behaviors not present in the original program (while retaining all original behaviors). In case several variables are assigned nondeterministically, not all combinations of assignments may be possible in the original program. On the other hand, nondeterministic assignment can reduce program complexity by masking relative execution speeds and program structure (as above). This makes it a vital tool in program verification.

### 1.6 Automata

The automaton is a fundamental structure in computer science, useful for much analysis and proof concerning "regular" sequential behavior. In the translation of a program to an automaton, defined next, the program's sequential state becomes the automaton "state", and the program's combinational states define automaton state transition conditions. Program behaviors are modeled by the sequences of combinational states consistent with successive automaton state transitions (*cf.* a discrete Markov process). The set of all such sequences form the *language* of the automaton, which provides the basis for model-checking as defined here.

To facilitate reduction and decomposition, the modeling automaton is "factored" into component automata, in emulation of the program's modularity. This factoring requires a means to correlate factor automata transition conditions that mediate the respective automaton state transitions. For example, if one program component reads the value of a variable set by another program component, the corresponding automata must have a mechanism to reflect a common view of the value of that variable. For reduction, there also must be a means to "abstract" transition conditions. If a state transition is conditioned on a variable `v` being nonzero, and the nonzero values of `v` are abstracted to the value "1" as above, then the corresponding automaton transition condition must be abstracted in a consistent well-defined manner. To meet these needs, the set of valuations of combinational program variables is given the structure of a Boolean algebra[1]. In this context, the correlation of transition conditions is captured by their conjunction in the Boolean algebra, and abstraction is obtained through a Boolean algebra homomorphism. The principal role of the automaton defined next is to serve as a "scaffolding" to carry this Boolean algebra: the factorization

[1] A *Boolean algebra* [8] is a set with commutative, associative, idempotent binary operations *conjunction* and *disjunction*, and the unary operation *negation*, which satisfy DeMorgan's Law; the set contains a universal element and its negation (called *zero*), and satisfies the natural relations for these. Every Boolean algebra can be represented by a set of subsets of some set, with the operations intersection, union, and set complement, and universal element consisting of the set. An *atom* is a nonzero element that is minimal in the sense that its conjunction with any nonzero element is itself or zero.

(4) needed for the decomposition (5) appears as a matrix tensor product on the automaton state transition matrices, and the simplification needed for reduction is given as a Boolean algebra homomorphism that acts on these matrices element-wise, in (11) below. The details are now explained.

In the context of program verification, the most useful type of automaton is the $\omega$-*automaton*, whose behaviors are (infinite) sequences. The automaton is defined in terms of a directed graph with a finite set of vertices called *states*, some of which are designated *initial*; a *transition condition* for each directed edge or *state transition* in the graph; and an *acceptance condition* defined in terms of sets of states and state transitions. Each *transition condition* is a nonempty set of elements called *letters* drawn from a fixed set called the *alphabet* of the automaton. There are several different equivalent definitions of acceptance conditions in use. The acceptance condition of an $\omega$-automaton can capture the concept of something happening "eventually", or "repeatedly" (*ad infinitum*), "forever", or "finally" (forever after). The set of all subsets of the alphabet forms a Boolean algebra $L$ in which the singleton sets are atoms. The set of atoms of $L$, denoted by $S(L)$, determines $L$ when $L$ is finite.

How are automaton transition conditions associated with the program's combinational states? If $v$ and $w$ are states of the automaton $A$, let $A(v,w)$ denote the transition condition on the directed edge $(v,w)$. Expressing conjunction (set intersection) in $L$ by $*$, we see for a letter $a$ that $a \in A(v,w)$ if and only if the atom $\{a\}$ satisfies $\{a\} * A(v,w) \neq 0$. In the context of $L$, we refer to the atom $\{a\}$ as a "letter", and $S(L)$ as the "alphabet" of $A$. We say the automaton $A$ is *over* $L$. $L$ is associated with the Boolean algebra generated by all valuations of combinational program variables: terms of the form $\mathtt{x} = c$ for a value $c$ in the range of the combinational variable $\mathtt{x}$. Thus, $L$ is all Boolean expressions in these terms. Each letter is a conjunction of terms of the form $x = c$, the conjunction over all the combinational variables $x$. This corresponds to the combinational state $\mathbf{C}$ with $x$-th component $c$. Thus, we associate combinational states with letters, *i.e.*, atoms of $L$. Automaton transition conditions are nonzero elements of $L$. For example, a transition condition $\mathtt{x} = 0$ corresponds to the set of all letters (*i.e.*, disjunction of atoms) of the form $\cdots * (\mathtt{x} = 0) * \cdots$. Referring to (1), we have

$$A(v,w) = \sum_{\mathbf{F}(v,\mathbf{C})=w} \mathbf{C}, \tag{2}$$

the summation expressing disjunction in $L$.

A sequence $(v_i)$ of states of $A$ is a *run* of a sequence of letters $(s_i) \in S(L)^{\mathbb{N}}$ provided $v_1$ is an initial state and for all $i$, $s_i * A(v_i, v_{i+1}) \neq 0$. A run is *accepting* if its infinitely repeating states and transitions satisfy the automaton acceptance condition.

The set of sequences of letters with respective accepting runs in an automaton $A$ is the *language* of $A$, denoted $\mathcal{L}(A)$.

$$\mathsf{decimal}(\mathsf{inputbits}) \neq \mathtt{output}. \tag{3}$$

The acceptance condition accepts runs that remain forever in the state $\mathtt{y} = 0$.

### 1.7 Program Factorization

The automaton model of a program is "built" from smaller automaton models of program components. A program component may comprise a single variable or a group of closely related variables. The program consists of its components. For example, the call-forwarding program may be implemented by respective components that define its setup and execution, as described earlier. A program is translated component-by-component to respective "component" automaton models. The component automata are combined as a "product" to form a single automaton that models the original program. This "factoring" of the program model into components follows the natural program structure, and aids in translation as well as in reduction and decomposition. All component automata are over one fixed Boolean algebra $L$ determined by the program.

The combinational variables of a program component may be interpreted as the "outputs" of that component. The simultaneous valuations of the outputs of a program component get translated to elements of $L$ that generate a subalgebra of $L$, the "output subalgebra" of the corresponding component automaton. (In this context, program primary inputs translate to outputs of trivial automata.) The call-forwarding setup component $p$ may have as outputs the variables $\mathtt{m}$ and $\mathtt{n}$ that designate the called and forward numbers, respectively. If there are no other outputs of $p$, then the output subalgebra of the automaton that models $p$ has as its atoms all expressions of the form $(\mathtt{m} = m_0) * (\mathtt{n} = n_0)$, for $m_0, n_0$ in the range of $\mathtt{m}$ and $\mathtt{n}$, respectively. The (interior) product of the respective output subalgebras of all the component automata is $L$. Each letter (atom of $L$) is a conjunction of atoms from the respective output subalgebras.

A program component is modeled by an *$L$-process* $P$: an $\omega$-automaton over the Boolean algebra $L$ with an identified "output" subalgebra $L_P \subset L$ that models the combinational states of the program component. The atoms $S(L_P)$ of the subalgebra $L_P$ are the "output values" of $P$. By (2), the output values $\mathbf{f}(v)$ represented above (*cf.* (1)), possible at the state $v$ of $P$, are the elements of $S(L_P)$ that have nonzero conjunction with the transition condition of some state transition leaving $v$. (The "inputs" to $P$ are the simultaneous collective output values of the various $L$-processes, *i.e.*, the alphabet $S(L)$. The transitions of $P$ may be independent of any particular outputs, of course.)

If $P$ and $Q$ are $L$-processes modeling respective program components $p$ and $q$, the *product* $P \otimes Q$ is an $L$-process that models the program component formed by taking $p$ and $q$ together. (This sometimes is called the "synchronous product" of $p$ and $q$, subsuming their "asynchronous" or "interleaving" product [9].) The set of states of $P \otimes Q$ is the Cartesian product of their respective state spaces, and the output subalgebra of the product is the (interior) product of the component output subalgebras.

If $P(v, w)$ is the transition condition for the transition $(v, w)$ from state $v$ to state $w$ of $P$ and $Q(v', w')$ is likewise for $Q$, then the transition condition for the transition $((v, v'), (w, w'))$ of the product $P \otimes Q$ is $P(v, w) * Q(v', w')$. Some program variables that define a component automaton's transition conditions may come from program components other than the one modeled by

the automaton. This allows automata to share conditions on the sequences they accept, and reflect coordination among the program components. For example, the call-forwarding execution component modeled by $L$-process $Q$ refers to the setup component's output variable `n` that designates the ultimate call destination number. In this way, $Q$ coordinates with the setup established by the setup component. The products of their respective transition conditions define the coordinated behavior of the two automata. If $C, D \in L$, then the product of the setup transition condition $P(v, w) = (\mathtt{n} = n_0) * C$ and $Q(v', w') = (\mathtt{n} = n_1) * D$ is nonzero only if $n_0 = n_1$. Thus, the transition condition $(P \otimes Q)((v, v'), (w, w'))$ of the product automaton is nonzero only when the two numbers agree.

The coordination thus defined by automata transition conditions supports the automaton "factoring" mentioned earlier. For any $L$-process we define its *state transition matrix* to be the matrix over $L$ whose $ij$-th element is the transition condition for the state transition from the $i$-th state to the $j$-th state. By the above definition of process product, the matrix for the product process $P \otimes Q$ is the tensor product of the matrices for $P$ and $Q$.

In general, if $M$ is the state transition matrix for an $L$-process that is factored into component $L$-processes with respective matrices $M_i$, then

$$M = M_1 \otimes \cdots \otimes M_k. \tag{4}$$

Moreover, if each process is designated by its respective matrix,

$$\mathcal{L}(M) = \mathcal{L}(M_1) \cap \cdots \cap \mathcal{L}(M_k). \tag{5}$$

The intuition for (5) is that each component $M_i$ imposes certain restrictions on the behaviors of the product $M$, and the behaviors of $M$ are the "simultaneous solutions" for behaviors of the respective components. If $M_1$ restricts the setup of the call-forwarding number `n` to numbers in a specified area code, and $M_2$ permits execution only if the designated number `n` is not busy, then the product pertains to numbers `n` in the specified area code that are not busy.

### 1.8 Model-Checking

We model as respective $L$-processes the program, its components, and the property to be verified. The program model is the product of its components (4).

It is convenient to use the same symbol to denote a process and its matrix, and henceforth the term *program* will be used to designate both the syntactic computer program and the $L$-process automaton model into which it gets translated.

Our formal definition of verification of property $P$ for a program $M$ is the automaton language containment check

$$\mathcal{L}(M) \subset \mathcal{L}(P).$$

In words, this says that all behaviors of the program are behaviors "consistent with" (*i.e.*, "of") the property. This is equivalent to checking that

$$\mathcal{L}(M) \cap \mathcal{L}(\widehat{P}) = \emptyset$$

where $\widehat{P}$ is the "complementary" automaton satisfying $\mathcal{L}(\widehat{P}) = S(L)^{\mathbb{N}} \setminus \mathcal{L}(P)$. By (5), verification is transformed into the automaton language emptiness check

$$\mathcal{L}(M \otimes \widehat{P}) = \emptyset. \tag{6}$$

The verification algorithm checks whether any state $(\mathtt{x}, 1)$ ever can be reached through a succession of state transitions that begins at the initial state $(0, 0)$. The analysis concludes when either a violation of the property is found (*i.e.*, $(\mathtt{x}, 1)$ is reached), or all reachable pairs $(\mathtt{x}, \mathtt{y})$ have been examined. The latter entails an examination of $2^{32}$ states, which could be costly were it not for a symbolic technique described below.

The foregoing is an automata-theoretic formulation of model-checking. There are other formulations of model-checking, expressed in terms of "temporal" logic formula satisfiability [7]. Each of these formulations can be transformed into (6) for some class of automata. In some cases, the best way known to perform the check is first to construct an automaton corresponding to the temporal logic formula and then to check (6). See [4,7,10] for a way to express the original and widely practiced form of model-checking for the logic CTL, without reverting to automata. The automata-theoretic formulation of model-checking was developed with a somewhat different perspective from the one presented here by M. Y. Vardi and P. Wolper [12] in the early 1980s, and by this author independently around the same time.

## 2 Algorithms

A general method to check whether $\mathcal{L}(A) = \emptyset$ for (6) is based on the finite structure of $A$. $\mathcal{L}(A)$ is nonempty (and the required property fails) if and only if $A$ has an accepting run. This is equivalent to the graph of $A$'s having a cycle "consistent" with the acceptance structure, since $A$'s state space is finite. The path from an initial state to such a cycle may be retraced, and this "error track" provides a step-by-step account of how the property can fail in the program.

One way to check for an accepting run involves an explicit enumeration of the states of $A$ reachable from an initial state. Since the number of states can be exponential in the size of the program description (4), explicitly enumerating the states has severe computational limitations. Fortunately, explicit enumeration is not necessary.

In 1986 R. Bryant made a seminal observation that played a major role in spurring the commercialization of model-checking techniques—a role that resulted in the ACM Kanellakis Prize for Bryant and his colleagues Clarke, Emerson, and McMillan, who showed how to use this beneficially for model-checking.

Bryant's observation was that binary decision graphs, long used for planning and programming, could be viewed as automata and thus minimized in a canonical fashion. The minimized structures were dubbed "binary decision diagrams" or BDDs. The idea was to represent the global state as a binary vector and represent a set of states by its characteristic function (having value 1 for each state

in the set), encoded symbolically as a Boolean function. The set of reachable states could be represented as a fixed point of the state set transformer that starts with the set of initial states and repeatedly applies the state transition matrix as an operator on the state space, adding the new states reached in one transition, from the set of states reached thus far. Since the set of states is finite, iterating this transformation has a least fixed point—the set of reachable states—in the lattice of state sets. Each iteration is expressed symbolically in terms of the Boolean characteristic functions, reduced to their canonical form and represented as a BDD. A very simple Boolean function can express an arbitrarily large set of states. For example, $x_1 = 0$ defines the set of all global states whose first bit is 0, representing half of the global states, no matter how large the state space. Thus, there is the potential to manipulate very large sets of states. In practice, the ability to compute reachable state sets with $10^{10}$ states for typical commercial programs is considered fairly trivial, $10^{20}$ states is routine and $10^{50}$ states and higher is not unusual. While the worst-case complexity of performing state reachability symbolically is the same as for explicit enumeration of the states, symbolic search lowered the threshold of acceptability for model-checking, leading to its commercialization.

### 2.1 Homomorphic Reduction

A program modeled by an $L$-process $M$ can be "abstracted" by a "simpler" program (with fewer variables or variables with smaller ranges) that is modeled by an $L'$-process $M'$. The relationship between $M$ and $M'$ is given by a Boolean algebra homomorphism $\phi : L' \to L$. If $\phi M'$ is the $L$-process with transition matrix obtained by applying $\phi$ element-wise to the transition matrix of $M'$, and $\mathcal{L}(M) \subset \mathcal{L}(\phi M')$ then we say $M'$ is a *reduction* (or "abstraction") of $M$, and $M$ is a *refinement* of $M'$. If this is the case and moreover,

$$\mathcal{L}(M') \subset \mathcal{L}(P') \tag{7}$$

for a property defined by the $L'$-process $P'$, then applying $\phi$ to both sides gives $\mathcal{L}(\phi M') \subset \mathcal{L}(\phi P')$. So for $P = \phi P'$, it follows that

$$\mathcal{L}(M) \subset \mathcal{L}(P), \tag{8}$$

which means that verifying the reduction verifies the refinement.

One type of reduction, *localization reduction* described below, is derived by an algorithm. So $M'$ and $\phi$ are defined algorithmically, and $\mathcal{L}(M) \subset \mathcal{L}(\phi M')$ is guaranteed by construction. Alternatively, a reduction can be "guessed" and then verified as above, where the "guess" consists of a definition of $M'$ and $\phi$. As an example of the latter, imagine a component adder used in the context of a program for an integrated circuit that computes the norm of an incoming signal. The correctness of the adder can be established by considering it in isolation from the rest of the circuit. $M'$ in this case could be just the model of the adder, with $\phi$ mapping the adder to the full circuit with all nonadder variables assigned nondeterministically.

In this example, a failure of the isolated adder does not necessarily imply it would fail in the context of the full circuit. If the otherwise-correct adder had errors in case of negative inputs, but negative inputs were impossible in the context of the full circuit, then the failure for negative inputs would not matter—and this "error" in fact may reflect an intentional optimization. In this case, one could constrain the inputs to the adder to be positive and prove it correct in that context. This approach would create a "proof obligation" with respect to the remainder of the circuit: to verify that the full circuit does in fact produce only positive inputs to the adder. In discharging such proof obligations, one must beware of circular reasoning: "the adder is good if the rest of the circuit offers it only positive inputs; the rest of the circuit offers only positive inputs if the adder is good" (perhaps a faulty adder gives feedback to the rest of the circuit that can cause the rest of the circuit to offer negative inputs).

In the earlier example of a "complex computation", all nonzero values of the program variable `v` were abstracted by the value "1", giving another example of a homomorphic reduction. The homomorphism maps $\mathtt{v} = 1$ in the abstract Boolean algebra to the disjunction of all nonzero assignments to `v` in the refined Boolean algebra.

In order to verify (8), the reduction (7) may be simplified by decomposing the property $P$ into small "subproperties" $P_1, \cdots, P_k$, where

$$\mathcal{L}(P_1 \otimes \cdots \otimes P_k) \subset \mathcal{L}(P). \tag{9}$$

The required check (8) is replaced by

$$\forall i, \quad \mathcal{L}(M) \subset \mathcal{L}(P_i).$$

By (5), this implies (8). Although one check is replaced with $k$, (8) may entail $O(|M|)$ computations for a model $M$ with $|M|$ sequential states, whereas each of the $k$ component checks can engender a reduction (7), with $P_i$ in place of $P$, that entails only $O(|M|^{1/k})$ computations. The check (9) can be recursively decomposed into a tree of such checks. At each node of the tree, the respective $k$ is small and the $P$ is checked against the product of the $P_i$'s at the successive nodes [9]. If the total number of nodes is $O(\log |M|)$, *i.e.*, is proportional to the number of program components, then the complexity of program verification is proportional to the size of the program. In some cases, when $P$ refers to a "repetitive" structure in $M$ like an array, $P$ can be decomposed algorithmically by restricting it to each successive array element. In general, a decomposition is obtained by a manual "guessing" step, and the guess is verified algorithmically as above.

Reduction and decomposition apply more generally to infinite-state program models. The principles behind them are central to most program verification methodologies that scale tractably with increasing program size.

*Design by refinement* is an application to verification of a general "top-down" program design methodology that has been around for many years [6,13]. By evolving a design together with its verification, a development methodology that leads to a tractable verification, as above, may be built into the design process.

Program details are added incrementally through a succession of program refinements:

$$\mathcal{L}(M_{i+1}) \subset \mathcal{L}(\phi_i M_i). \tag{10}$$

Here, the increasing "level" $i$ indexes progressively more refined program models, leading ultimately to the program with all its details elaborated. Since $M_i$ is a reduction of $M_{i+1}$, any property verified for $M_i$ holds for $M_{i+1}$, and on. Properties of the program thus may be verified once and for all at the most abstract level possible, and each level may be verified before the next level is defined.

Based on the composition (4) of each level $M_i = M_{i1} \otimes \cdots \otimes M_{ik_i}$, the check (10) can be decomposed into a set of smaller sufficient checks: for each $j$,

$$\mathcal{L}(M_{(i+1)h_1} \otimes \cdots \otimes M_{(i+1)h_t}) \subset \mathcal{L}(\phi_i M_{ij}), \tag{11}$$

where the factors $M_{(i+1)h}$ in the successive checks (11) are factors of $M_{i+1}$.

### 2.2 Localization Reduction

The *variable dependency graph* of a program is the directed graph whose vertices are the program's variables, with an edge $(\mathtt{v}, \mathtt{w})$ whenever $\mathtt{v}$ appears in the program's assignment expression for $\mathtt{w}$. An automatic way to derive a homomorphic reduction involves a traversal of the variable dependency graph, to determine which values of which variables are equivalent, with respect to the property being checked. A variable $\mathtt{v}$ is *irrelevant* if it has no directed path to the variables that implement the automaton $P$ that defines the property being checked. In this case, if we transform the program's acceptance conditions to $P$, the particular values assigned to $\mathtt{v}$ can have no bearing on the check (6). Two values of a variable are *equivalent* if the assignment expressions of the relevant variables do not distinguish them. A homomorphism may associate together all equivalent values.

Localization reduction is an iterative algorithm that starts with a small "active" program component that is topologically close in the variable dependency graph to $P$ (Figure 2). All other program variables are abstracted with nondeterministic assignments. This renders the values of the variables beyond the boundary of the active variables equivalent, so operationally these variables may be "pruned" out of the model. If the property is thus verified, then by (5) it holds for the full program. On the other hand, if the property fails for this reduction, the algorithm attempts to expand the resulting error track to an error track for the full program. If this succeeds, it means the error in fact reveals a "bug" in the full program. If, however, the expansion fails, it means the error is an artifact of the localization. In this case, the active component is augmented by a heuristically small set of topologically contiguous relevant program variables whose assignments are inconsistent with (and thus invalidate) the error track. The verification check is repeated for the expanded program component, and the process is iterated until the algorithm terminates with a verification or a genuine program error.

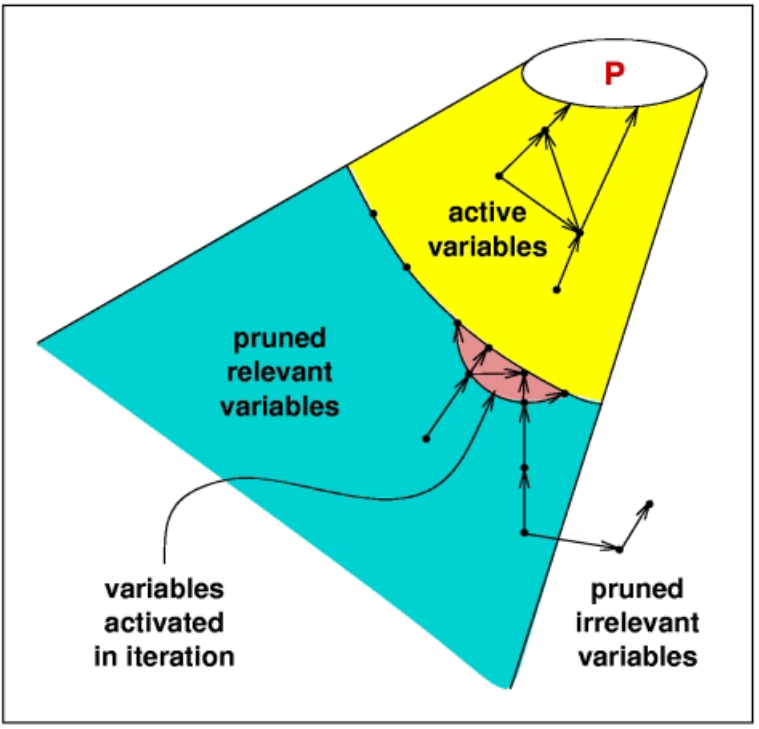



**Fig. 2. Localization Reduction.** In the variable dependency graph, a program component close to the automaton $P$ that defines an aspect of the program's required operation is designated *active*. The remaining variables are *pruned*, removing their effect. If the required operation is verified for this reduction, it holds for the full program. If it neither verifies nor falsifies, the active component is augmented, and the check is repeated.

## 3 Conclusion

Program verification can increase the reliability of a program by checking it in ways that may be overlooked in conventional testing. Conventional testing intrinsically comes at the end of the design cycle, whereas program verification may be introduced at the beginning, eliminating bugs earlier than otherwise possible. This can accelerate program development.

A principal type of program verification is model-checking, which may be expressed in terms of automata. Although the worst-case time complexity for model-checking is for all practical purposes exponential in the size of the program to be verified, model-checking often can be accomplished in time proportional to the size of the program, as follows. Define the properties to be checked; decompose each property into subproperties that admit of respective tractable reductions; verify each subproperty on its respective reduction. This approach is "bottom-up", since it begins with the full program.

The alternative "top-down" approach starts with the same property decomposition, but before the program exists. Rank the subproperties according to level of abstraction. For each level, define an "abstraction" of the program-to-be. This abstraction corresponds to a reduction in the bottom-up approach. Program details are added incrementally, ending with the fully elaborated program. Each increment is verified to be a refinement of the previous level. A property verified at one level thus remains true at all subsequent levels. Since the program is written and checked incrementally, debugging starts earlier in the program development cycle than with conventional testing, which requires the fully elaborated program. With the top-down approach, the program can be designed so that each required check is tractable.

Reductions can be derived algorithmically in the bottom-up approach. In either approach, a prospective reduction may be verified to be an actual reduction by checking the refined program against a homomorphic image of the reduced program.

For both the bottom-up and top-down approaches, property decomposition is a fundamental step. In special cases, when a property refers to a repetitive structure, its decomposition can be derived algorithmically. In general, decomposition can be verified—but not derived—by a tractable algorithm. In fact, it is not tractable to determine whether a "good" decomposition—one that gives rise to tractable bottom-up verification—exists. Finding useful heuristics for decomposition is a foremost open problem in model-checking.

## References

1. J. Barwise, Mathematical proofs of computer system correctness, *Notices Amer. Math. Soc.* **36** (1989), 844–851.
2. W. W. Bledsoe and D. W. Loveland (eds.), *Automated Theorem Proving: After 25 Years*, Contemporary Math., vol. 29, Amer. Math. Soc., Providence, 1984; especially R. S. Boyer and J. S. Moore, Proof-checking, theorem-proving and program verification, pp. 119–132.
3. E. M. Clarke and E. A. Emerson, Design and synthesis of synchronization skeletons using branching time temporal logic, *Logic of Programs: Workshop, Yorktown Heights, NY, May 1981*, Lecture Notes in Computer Science, vol. 131, Springer-Verlag, 1981.
4. E. M. Clarke Jr., O. Grumberg, and D. Peled, *Model Checking*, MIT Press, 1999.
5. R. DeMillo, R. Lipton, and A. Perlis, Social processes and proofs of theorems and programs, *Comm. ACM* **22** (1979), 271–280.
6. E. W. Dijkstra, Hierarchical ordering of sequential processes, *Acta Informatica* **1** (1971), 115–138.
7. E. A. Emerson, Temporal and modal logic, *Handbook of Theoretical Computer Science*, vol. B, Elsevier, 1990, pp. 995–1072.
8. P. Halmos, *Lectures on Boolean Algebras*, Springer-Verlag, 1974.
9. R. P. Kurshan, *Computer-aided Verification of Coordinating Processes—The Automata-Theoretic Approach*, Princeton University Press, 1994.
10. K. L. McMillan, *Symbolic Model Checking*, Kluwer, 1993.
11. S. Schroeder, Turning to formal verification, *Integrated System Design Magazine*, Sept. 1997, 1–5.

12. M. Y. Vardi and P. Wolper, An automata-theoretic approach to automatic program verification, *Proc. (1st) IEEE Symposium on Logic in Computer Science*, (1981), 322–331.
13. N. Wirth, Program development by stepwise refinement, *Comm. ACM* **14** (1971), pp. 221–227.

# Reformulation in Planning

Derek Long, Maria Fox, and Muna Hamdi

Department of Computer Science
University of Durham, UK
d.p.long@dur.ac.uk maria.fox@dur.ac.uk

**Abstract.** Reformulation of a problem is intended to make the problem more amenable to efficient solution. This is equally true in the special case of reformulating a planning problem. This paper considers various ways in which reformulation has been exploited in planning. In particular, it considers reformulation of planning problems to exploit structure within them by allowing deployment of specialised sub-solvers, capable of tackling sub-problems with greater efficiency than generic planning technologies. The relationship between this reformulation of planning problems and the reformulation of problems in general is briefly considered.

## 1 Introduction

The reformulation of a problem is intended to make the problem more amenable to efficient solution. While problems can usually be expressed in many ways, it is often the case that a particular problem-solving strategy is applicable only to problems expressed in a certain form. In this case, reformulation is the means by which a useful strategy can be brought to bear on a problem — the problem is reformulated in the canonical form in which that the particular strategy can be applied. Similar benefits can be obtained when the original problem expression is already in a form that can be tackled by a strategy, but reformulation can allow the strategy to be applied more effectively. Both of these situations can be seen as cases within the scenario depicted in figure 1. The figure illustrates how reformulation of a problem can allow different elements (or strategies) within a problem-solving system to be brought to bear on a problem by reformulating it. The figure illustrates that reformulation of a problem can allow different problem-solving strategies to be applied to it. A special case is where reformulation can allow the same strategy to be applied but in a more efficient way. The figure also suggests an important point: a single problem might require multiple strategies to solve it and reformulation might change the combination of strategies that can be applied to solving the problem.

In this paper we consider the role of reformulation in planning, examining several of the ways in which it has been exploited. We then turn to an important use of reformulation based on our notion of *generic types* and discuss how this approach can be used to improve planner performance. We also consider how generic types can be used to support alternative approaches to reformulation and, finally, discuss how the strategy that we have identified, for applying reformulation to planning, might generalise to other problems.

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 18–32, 2002.

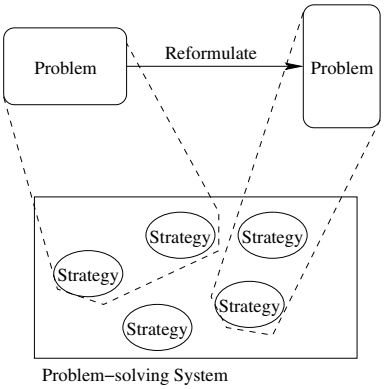



**Fig. 1.** Reformulation and redeployment: reformulating a problem can allow different problem-solving strategies to be deployed to solve it.

## 2 Reformulation in Planning

Planning problems are no exception to the possibility of benefits from reformulation. If we consider the problem-solving system in figure 1 to be a loose collection of strategies that might include constraint satisfaction strategies, general planning strategies, SAT-solvers and other, more specialised, problem solvers, then there have already been several efforts at reformulation of planning problems described in the literature. For example, Kautz and Selman have considered reformulating planning problems as SAT-problems in order to apply SAT-solving strategies to them [25]. Van Beek showed how planning problems can be reformulated as finite-domain constraint satisfaction problems and a CSP solver applied to solve them [39]. Where van Beek reformulated the problems by hand, Kambhampati and Binh Do have shown that automatic reformulation to CSPs is possible [10], allowing automatic redeployment of a planning problem from a generic planning strategy (such as Graphplan [4] or FF [24]) to a CSP-solver. Again, Cimatti, Roveri and Traverso have shown that planning problems can be reformulated as model-checking problems and an OBDD strategy applied to them [6]. All of these pieces of work have relied on a complete reformulation of planning problems from a classical action-based planning domain representation into the forms appropriate for each of the respective general problem-solving strategies (SAT-solving, CSP-solving or model-checking). This can be seen as the replacement of one "pure" problem-solving strategy with another.

The benefits of such a wholesale replacement of one generic problem formulation into another are sometimes rather ambiguous: the objective is often to take advantage of a well-developed strategy, perhaps one having made recent gains in performance. For example, the early work on reformulation to SAT-problems took advantage of then recent advances in SAT-solving technology to seek performance gains in planning. However, planning by SAT-solving has not demonstrated a convincing long-term benefit (no SAT-solver planning technology has demonstrated challenging performance compared with the current leading generic purpose-built planning technology such as FF [24], HSP2 [5]

or the more recent LPG [23]). Planning by model-checking has also not shown convincing performance benefits and the significant number of model-checking approaches participating in the 2nd International Planning Competition in 2000 has dropped to there being no examples of pure model-checking in the 3rd (and most recent) competition although MIPS [12] adopts a hybrid approach which includes a model-checking component. CSP-solving has been rather more successful as a strategy for planning by complete reformulation of planning problems, with Sapa [9] showing some promising performance in the 3rd IPC.

In the context of the competitions, at least, application of reformulation is constrained to be via fully automatic reformulation, unless one considers the "hand-coding" planners (such as TLPlan [3], TALPlanner [27] and SHOP2 [34]), for which domains are recoded by hand, to be exploiting reformulation. This would be somewhat misleading, since the recoding of the domains for these planners is not simply to reshape the domain, but to allow control rules to be added to the encoding that control the search in the planning systems. Although one can consider the addition of new information as part of the process of reformulation, it is clearly a far more expensive and knowledge-intensive task if this is intended and the subsequent benefits in problem-solving performance are therefore bought at a significant price in the reformulation efforts. Considered as a complete approach to solving planning problems there remains controversy in the community about the extent to which planning by "reformulating" through the addition of knowledge-intensive control information represents a generally acceptable tradeoff between reformulation effort and performance gains.

### 2.1 Reformulation and Expressiveness

One benefit that reformulation can offer is the ability to tackle problems expressed in an enriched formalism, where the formalism can express problems that extend beyond the capabilities of existing solving strategies. For example, Gazen and Knoblock [21] collected, completed and formalised common techniques for conversion of various elements of the expressive power of the ADL [36] extensions of planning domain descriptions into the simpler STRIPS subset. This reformulation allows the simpler STRIPS planning strategies to tackle problems expressed in the richer language. The greater expressive power of the ADL extension can be seen in the cost of the reformulation, which can be exponential in the size of the problem encoding in certain cases. Thus, the extra expressive power allows an exponential compression relative to a STRIPS encoding and can therefore lead to exponentially worse performance from a STRIPS planning strategy that the size of the original ADL encoding might suggest should be expected. A more practical reformulation is applied in IPP [26] to handle ADL expressive power more effectively. In that system the use of conditional effects, responsible for the most common exponential blow-up in the encodings of problems in the scheme proposed by Gazen and Knoblock, is handled by an extension to a STRIPS planning strategy, leaving the remaining elements of ADL to be reformulated into simpler forms. The gain is that the treatment of conditional effects in IPP can often be managed efficiently, while the Gazen and Knoblock approach will always cause combinatorial growth in domain encodings. A similar approach is taken in SGP [2] and in the more recent FF. Nebel has shown how the relative expressive power of these language extensions can be compared more

formally, precisely by a reformulation technique, in [35]. In that work, Nebel considers the effect of reformulation of the original problem in such a way that a solution to the reformulated problem *allows recovery* of a solution to the original problem. This is, in fact, a common technique when applying reformulation approaches to problem-solving: a problem is reformulated and solved, if all goes as expected, more efficiently, and then the solution to the original problem is extracted from the solution to the reformulated problem.

Another example of the same approach is in Fox and Long's work on temporal planning in the system LPGP [17]. In that system, planning problems in which there are durative actions, which are actions with duration that can have conditions and effects attached to both their start and end points and invariant conditions attached to the duration of their activity, are reformulated as collections of simple non-durative actions. These actions, together with some important linking constraints, allow a relatively straightforward extension of a non-temporal Graphplan planning strategy to handle much more expressive temporal planning problems.

A reformulation of this kind could, in principle, also be used to tackle the use of numeric valued expressions within planning domains. Since a finite plan can only ever introduce finitely many new numeric values, which it is possible to identify by a finite reachability analysis (such as a plan-graph construction), a propositional planning strategy can be used to tackle problems involving numbers using a continual reformulation as the strategy considers longer and longer possible solutions in its search for a plan. Whether such a strategy could be useful in practice would depend on the number of numeric values introduced during reachability analysis.

Occasionally reformulation can be used to to exploit a more powerful solver more effectively, where a problem has been expressed using a simpler expressive power that the solver can exploit. A simple example is that in which the domain analysis system, TIM [13], can be used to infer types in an untyped planning domain description, enriching the domain description and allowing a system that can exploit type information to improve its performance accordingly. Another example is the use of TIM to identify symmetries in a planning domain [14,16], reformulating a problem in order to allow exploitation of symmetry elimination in the planning machinery. TIM and another fully automatic system, DISCOPLAN [22], both support reformulation of planning problems by the addition of mutex information and other constraints that can be exploited by planning systems.

## 3 Reformulation of Planning Problems for Deployment of Multiple Problem-Solving Strategies

The majority of the work described so far is directed at the reformulation of a planning problem into a form that can be tackled by a complete problem-solving strategy. Although this is obviously an important role for reformulation, it is also possible, as figure 1 suggests, to reformulate a problem in order to exploit multiple problem-solving strategies. An hypothesis that has driven an important line of research is that generic problem-solving strategies are unlikely to be the most efficient tools with which to tackle specific sub-problems that commonly arise as a part of larger problems. For ex-

ample, the general search approaches that underlie many planning and CSP solving systems are not ideally suited to tackling problems that involve finding efficient routes for agents moving around while executing a plan. Much more effective is a tool that can exploit the fact that the problem involves finding shortest paths, possibly including paths that visit specific locations in sequence, or as an unordered collection, and can use a problem-solving strategy that is tailored to that problem.

Several researchers have identified the benefits of specialised treatments of problems that arise naturally as an element of planning problems, particularly scheduling and resource handling [11,28,38,15]. In almost all of these systems the planning problem is reformulated by hand in order to allow the planner to identify the separation of the sub-problem or sub-problems from the remainder of the planning problem. The whole problem can the be deployed across multiple solvers, including, possibly, a generic planning strategy to be applied to the core of the planning problem left once the sub-problems have been separated. The separation of a *hard* problem (NP-hard or worse) from a planning problem offers far more hope for handling it effectively than attempting to use general planning technology. It is unlikely in the extreme that a general planning strategy can prove a powerful heuristic approach to managing, for example, combinatorial resource management problems, Travelling-Salesman-variant route planning problems, Job-Shop-Scheduling-variant resource allocation problems and so on. In IxTeT [28] resources are handled by reformulating planning problems to make explicit the resource-producing and resource-consuming actions and then by using a resource-constraint manager to handle the constraints on the resources used within developing solutions. Similarly, resource profiling is used in OPlan [11]. In RealPlan [38] the problem of scheduling transporters to cargoes is handled by a separate scheduler. In all of these systems the communication between sub-solvers (resource manager, profiler or scheduler) and the rest of the planning system is a sophisticated technical problem.

In [15] Fox and Long describe Hybrid STAN, a planning system in which a special purpose strategy for addressing route-planning sub-problems is integrated with a general planning system. This system is based on *automatic reformulation* of a planning problem from a standard action-based formulation into one in which the sub-problems are identified and linked to the remainder of the problem using specialised expressions associated with actions that rely on conditions established within the fragment of the planning problem that is identified as a sub-problem. In [15] the need for a more general form of interface between the planner and its sub-solvers is indicated, allowing the communication of constraints between all the processes participating in the solution of a diversely structured problem. In this paper we outline some of the progress we have made towards the development of such an interface. We describe the notion of *active precondition* — a general means by which information can be communicated between a planner and one or more sub-solvers. Active preconditions represent an expressive form that can be exploited in the reformulation of a planning problem from a pure action-based model into one that makes explicit the relationships between those actions and sub-problems within a planning problem. We proceed to discuss this process of reformulation. We then demonstrate the use of active preconditions in the integration of STAN with two specialised solvers: one for planning the routes to be followed by mobile objects committed to visiting various locations in a plan, and one for allocating drivers to

these mobiles. The domains we use for demonstrating the power of our integrated system feature mobiles that must be driven (in contrast to mobiles that are self-propelled, such as those in the Logistics domain).

### 3.1 Automatic Reformulation of Planning Problems through Generic Types

The automatic reformulation of planning problems rests on a well-established line of research that developed from the TIM system [13]. The development introduced the notion of a *generic type* [29] (the developments and the relationship to the original system can be seen in more detail in [32]).

A type, in a planning domain, is a set of objects that can all be used to instantiate the same subset of arguments of action schemas within the domain (although not all of the instantiated actions will necessarily be applicable, because of unsatisfied preconditions). Thus, types in planning domains are actually based on a functional commonality between objects within a single domain. In contrast, generic types are collections of types, characterised by specific kinds of behaviours, examples of which appear in many different planning domains. Thus, generic types are defined *across* domains, rather than within single domains. For example, domains often feature *transportation* behaviours since they often involve the movement of self-propelled or portable objects between locations. In the context of recognising transportation domains TIM can identify *mobile* objects, even when they occur implicitly, the operations by which they move and the maps of connected locations on which they move, the *drivers* (if appropriate) on which their mobility depends, any objects they can carry, together with their associated *loading* and *unloading* operations. The analysis automatically determines whether the maps on which the mobiles move are static (for example, road networks) or dynamic (for example, corridors with lockable doors). The recognition of transportation features within a domain suggests the likelihood of route-planning sub-problems arising in planning problems within the domain.

In addition to mobility-related generic types, other generic types have been identified and characterised. A generic type has been identified for *construction* behaviours [7]. Construction problems are commonly associated with iterative behaviour, suggesting the presence in the domain of types with inductive structure (analogous to lists and trees) associated with well-defined inductive operations. Recognition of these features supports an abstract level of reasoning about the domain. Generic types representing certain kinds of *resources* which restrict the use of particular actions in a domain have also been characterised [30]. The presence of these features suggest that processor and resource allocation sub-problems might arise and might be related to combinatorial sub-problems such as Multi-processor Scheduling or Bin Packing. TIM is able to recognise the existence, in a domain, of finite renewable resources which can be consumed and released in units [31].

The analysis performed by TIM takes as input a standard STRIPS or, following recent extensions [8], an ADL description of a domain and problem. Some experiments have also been conducted with domains using numbers [18]. It should be emphasised that the analysis is automatic: no annotations are required to identify special behaviours and the analysis is completely independent of the syntactic labels used to describe the objects and operations. As a consequence, the analysis can recognise generic behaviours

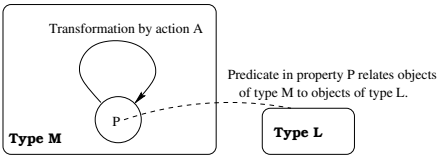



**Fig. 2.** A simple generic type fingerprint: the mobile type.

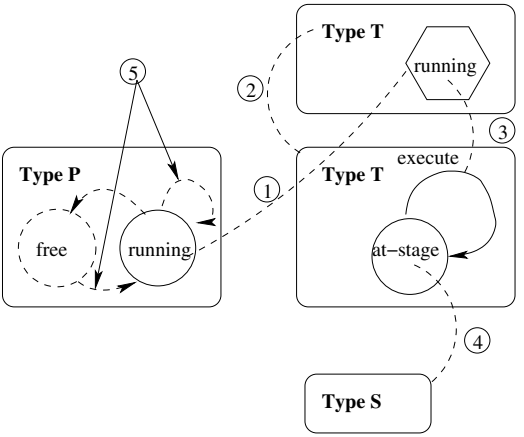
in domains which do not obviously fall into the categories indicated by the names of the generic types. This allows automatic reformulation of problems where a human domain-engineer might not identify the possibility. Recognition is based on the discovery of patterns within the structure of a domain encoding that can be described as *fingerprints* denoting the occurrence of specific behaviours amongst objects within the domain. The fingerprints are described in terms of relationships between finite-state machines that are extracted by TIM from a domain description and which show how objects in a domain make state transitions as actions are applied to them. For example, figure 2 represents the simplest pattern, indicating the existence of mobile objects. Figure 3 is an example of a more sophisticated pattern, this one identifying the existence of objects that display a particular constrained resource behaviour corresponding to a Multi-Processor Scheduling problem. In the figure the type $P$ is the processor type and a processor object must be allocated to a task (type $T$) instance before that task can execute (moving from stage to stage). An example of a domain encoding that displays this behaviour (in its most explicit form) is given in figure 4.

Although the recognition of different generic types is still carried out using *ad hoc* analysis techniques, based around the core TIM analysis, work is in progress towards the unification of these techniques based on a single matching strategy and a common framework for the description of generic types and relationships between them.

Once generic types have been identified in a domain, the planning problem can be reformulated in terms of sub-problems. A *sub-problem* is defined to contain the following components:

- A collection of objects capable of a specific behaviour (eg: trucks are capable of mobility);
- A collection of predicates capturing this behaviour (eg: *at* captures locatedness, *in* captures portability, *free* captures ability to be allocated to a task, etc).
- A collection of operators that affect these predicates (eg: *drive* affects *at*, *load* affects *in*, *allocate* affects *free*, etc).

TIM identifies the relevant objects, predicates and operators in a given domain. For example, figure 5 shows a particular encoding of the Multi-Processor Scheduling sub-problem, associated with the generic type of *processable task* (type $T$ in figure 3). In this example there are three different types of objects participating in the sub-problem, and predicates associated with each of the three types. These predicates are the critical ones for solving the MPS problem. The collection of operators contains operators responsible for changing the states of the objects with respect to these key predicates.




1: A type, P, contains a state defined by a predicate that links to a single instance of a type, T. The T instance acquires the corresponding property as an attribute.
2: The type, T, has a second property space, defining state−transition behaviour.
3: The attribute of T is an enabling condition for the transition, execute, in the second space for T.
4: The state for T is a property that links T to instances of another type, S.
5: The elements of P can enter the state in which they are related to elements of T by either a simple loop (reallocate) or by a transition (release) to a second state (free) and a transition back (allocate).

**Fig. 3.** The fingerprint of the generic type of driver objects.

```
(:action execute
        :parameters (?p ?j ?s ?t)
        :precondition (and (running ?p ?j) (stage ?j ?s) (nextto ?s ?t))
        :effect (and (stage ?j ?t) (not (stage ?j ?s))))

(:action swap-to
        :parameters (?p ?j)
        :precondition (and (idle ?p) (job ?j))
        :effect (and (running ?p ?j) (not (idle ?p))))

(:action swap-from
        :parameters (?p ?j)
        :precondition (and (running ?p ?j))
        :effect (and (idle ?p) (not (running ?p ?j))))
```

**Fig. 4.** Canonical Multi-Processor Scheduling (CMPS) as a Planning Domain.

**MPS Sub−problem**

| | |
|---|---|
| Objects: | *tasks, stages, processors* |
| Predicates: | *free, allocated, priocessed* |
| Operators: | *allocate, de−allocate, re−allocate, process* |

**Fig. 5.** An encoding of the MPS Sub-problem

The reformulation of the planning problem links sub-problem structures to actions by a process of abstraction. Abstraction of a recognised sub-problem from the planning domain involves the removal from the domain description of the sub-problem operators and the modification of all remaining operators to remove reference to the sub-problem predicates. In this way, all references to the sub-problem are removed from the planning domain description, leaving a *core* problem for the planner to solve. In [15] is described a simple form of integration between a forward planner and a route-planning sub-solver. This integration is based on passing information from the planner to the sub-solver about the current and required locations of mobiles that are required to move. The interface mediating the interaction between the planner and sub-solvers described in that work has now been generalised into a more powerful form of interface called an *active precondition*.

We begin by defining the structure of an *object specification*:

**Definition 1** *An* Object Specification*, $OS_t$, associated with an object in a planning problem, $t$, is a triple containing the following components:*

1. *The current state of $t$. This is initialised from the initial state and is updated every time $t$ changes its state;*
2. *The final state of $t$, taken from the goal specification. If $t$ has no state specified in the goal destination this field is null.*
3. *The generic structure of $t$. This defines what kinds of generic behaviours $t$ supports.*

The state of an object is the collection of properties that refer to that object in the current state of the world. Typically, the state of an object can be partitioned into collections of propositions that identify the state of the object relative to one or other of the generic behaviours the object supports. For example, a mobile object will support movement between locations and the current location is its state relative to this behaviour.

**Definition 2** *An* Active Precondition*, $AP_o$, with respect to a set of objects $os$, is a triple containing the following components:*

1. *A proposition, $P$, involving $os$;*
2. *The object specifications, for each $o \in os$, $OS_o$;*
3. *The identity of the sub-solver that is responsible for satisfying $P$.*

The active precondition is a data structure that couples the original precondition of an action to the objects that are involved in the proposition and the sub-solver that can bring about the necessary condition. Often the objects that appear in a proposition are not all equally significant. For example, if the proposition is that a certain task must reach a particular stage then the task is the more important object since it is the task that must be processed in order to reach a particular stage. The precise significance of each object in an active precondition depends on the way that the sub-solver manages the process of achieving the corresponding proposition.

If we consider a planning problem that includes an instance of the generic type of processable tasks, then we can see that the object specification for a task will include the information indicating what stage the task is currently at, what stage it is to reach and whether it currently has a processor allocated to it or not. An object specification is a

data structure that encapsulates the representation of the state associated with the object. The advantage of this is that, having identified a particular instance of a generic type to which the object belongs, we can select a representation for the object specification that makes it more efficient to access the corresponding properties of the object. In the case of the task, we can store its current stage, its goal stage and whether or not there is a processor allocated to it in an internal form that is much more easily modified as the task is processed than a simple set of propositions would be.

Figure 6 shows how this abstraction is achieved with a task argument to a packing action. It can be seen that each action that has a precondition that is to be solved by a sub-solver will acquire a collection of active preconditions to replace them, along with the remaining standard preconditions. Each active precondition is responsible for handling only one proposition, but an action can have many active preconditions.

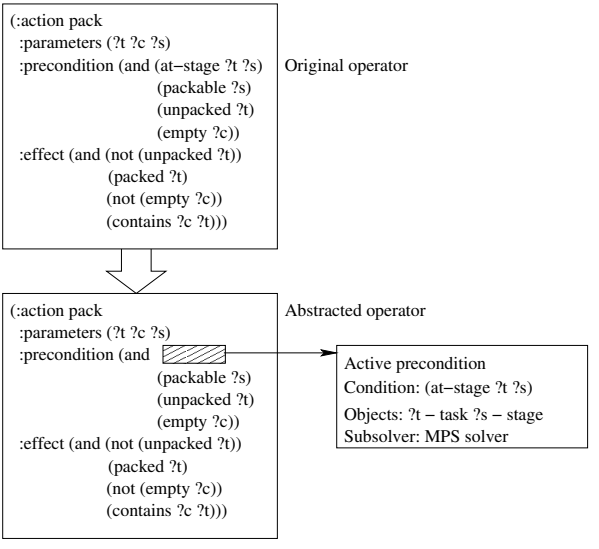



**Fig. 6.** The abstraction process for a packing operator. The *at-stage* precondition for the *task t* is replaced with an active precondition.

In addition to replacing the preconditions with active preconditions in this way, the abstraction process will remove the action responsible for the generic movement behaviour associated with the mobile type. In certain cases the process of asbtraction can leave a null domain description (this happens in the canonical MPS problem encoded as a planning problem [30], where there is no other structure than that of the MPS sub-problem), but in general there remain components of the original problem that have to be solved by the planner. Active preconditions form one component of the mechanisms by which sub-solvers communicate. The way in which they are currently exploited is described in the next section.

### 3.2 Exploitation of Generic Types

In order to exploit the reformulation, our planning system is driven by a forward search engine that attempts to solve the remaining planning problem (if there is one), using a relaxed-plan heuristic in much the same way as FF [24]. As relaxed plans are constructed in order to evaluate alternative state transitions these relaxed plans create agendas, formed from the active preconditions of actions selected in the relaxed plan. The agendas are then examined by sub-solvers to identify the goals that have been posed that fall under the remit of each of the sub-solvers. The sub-solvers then communicate to the forward planner an estimate of the cost of achieving the corresponding goals within the sub-problem for which they are responsible. Using this enriched goal distance estimate the forward planner selects the best transition by which to progress.

If the action selected contains an active precondition then the corresponding sub-solver is given a new task: to construct an actual plan fragment which will achieve the necessary precondition from the current state prior to insertion of the newly selected action. The abstraction process ensures that all of the materials required by the sub-solver in order to achieve this are under the control of the sub-solver, so that the goal can be achieved. This process ensures efficient solution of the sub-problem and can utilise global state information about the progress of the solution in order to ensure that resources are sensibly deployed — for example, the MPS solver can ensure that loads are balanced between processors in order to achieve an efficient solution.

There are complications in this linkage that we have not space to address properly in this paper. An initial report of some of the progress we have made in handling the difficulties that arise when sub-problems are interdependent can be found in [19]. In particular, we have identified the relationships that can exist between sub-solvers and their respective responsibilities and can automatically build a dependency network that allows these relationships to be identified. Armed with this information it is possible to decompose the agenda constructed by a planner so that sub-solvers examine it in an order that allows sub-solvers to impose further goals for subsequent sub-solvers to achieve. There remain several significant challenges to resolve in order to make this approach fully general.

To give a broad idea of what can be achieved using the reformulation into sub-problems we present one data set. Data demonstrating performance in other problem sets can be found in, for example, [15,7,19]. The current data set shows performance on a collection of randomly generated MPS problems.

In this test we compared IPP [26], BlackBox [25], FF [24] and STAN on CMPS (the canonical MPS problem illustrated earlier) instances involving increasing numbers of processors and tasks. We were interested in comparing both time taken to solve the instances and makespan of the solutions. All experiments were performed on a Celeron 333 Intel processor and a machine with 256Kb of RAM, 256Kb swap space. The larger problems defeated FF primarily in the instantiation phase. It should be noted that the reformulation being carried out by TIM in these problems includes the addition of type information which is only impicit in the encodings. The instantiation by FF is improved if type information is explict, but even with this information FF cannot solve the largest half-dozen problems on this machine. The quality of the plans produced by FF is consistently poor, with all tasks being allocated to a single processor until late in the sequence when

| Procs | Tasks | IPP | | Blackbox | | FF | | STAN | | Diff | Sub |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Secs | Span | Secs | Span | Secs | Span | Secs | Span | | Opt? |
| 2 | 5 | 3.28 | 11 | 12.72 | 11 | .01 | 19 | .006 | 11 | 0 | |
| 2 | 10 | | | | | .1 | 72 | .008 | 38 | 1 | |
| 2 | 20 | | | | | .87 | 233 | .066 | 118 | 0 | |
| 3 | 10 | | | | | 1.49 | 64 | .088 | 25 | 1 | |
| 3 | 20 | | | | | 1.83 | 211 | .064 | 79 | 1 | |
| 3 | 30 | | | | | 8.92 | 477 | .073 | 170 | 1 | |
| 4 | 20 | | | | | 53.07 | 215 | .017 | 59 | 1 | |
| 4 | 30 | | | | | | | .09 | 128 | 2 | 1 |
| 4 | 40 | | | | | | | .121 | 190 | 2 | |
| 4 | 50 | | | | | | | .141 | 358 | 3 | 1 |
| 10 | 50 | | | | | | | .206 | 145 | 10 | 2 |
| 10 | 80 | | | | | | | .560 | 300 | 3 | |
| 10 | 100 | | | | | | | .954 | 548 | 2 | |
| 15 | 50 | | | | | | | .132 | 97 | 6 | 2 |
| 15 | 100 | | | | | | | .98 | 366 | 4 | |
| 15 | 150 | | | | | | | 2.48 | 806 | 4 | |
| 20 | 50 | | | | | | | .153 | 72 | 3 | |
| 20 | 80 | | | | | | | .504 | 150 | 10 | |
| 20 | 100 | | | | | | | .894 | 275 | 5 | |
| 20 | 150 | | | | | | | 2.498 | 606 | 3 | |

**Fig. 7.** Results for MPS domain instances. Instances were randomly generated.

one or two alternative processors are sometimes used. This is only to be expected, since FF generates plans that IPP and BlackBox both found the optimal parallel plan for the smallest instance but had insufficient memory to solve any further instances. BlackBox attempts first to use a GraphPlan strategy and only switches to SAT-solving if this has failed to make progress when a fixed time cut-off is reached. The first instance was small enough to be solved before the cut-off so BlackBox used its Graphplan strategy in this case.

STAN is able to solve all of the instances without any search at all. When TIM has identified the MPS nature of the problem a heuristic strategy is invoked to solve the sub-problem. The first-fit decreasing heuristic produces very good quality solutions. The Diff column in the table shows the difference in load between the lightest and heaviest loaded processors. As can be seen, the difference is generally very small indicating that all processors are being allocated an approximately average load. The final column shows the extent to which STAN's solutions seem likely to deviate from optimal. By inspection of the proposed allocation it can be determined whether it is, in principle, possible to rearrange the loads to result in a shortened makespan. Whether this is actually possible or not cannot easily be determined (we would need to search for an alternative solution or compare our solutions with ones generated by alternative heuristics or approximation schemes). Only four of the solutions were possibly non-optimal and these only by very small margins.

## 4 Generic Types and Reformulation

So far we have discussed the use of generic types in their role as a foundation for automatic reformulation of planning problems, based on automatic recognition of fingerprints within action-based models. A more recent direction of work is exploring the use of generic types as *planning domain design patterns*, analogous to the software engineering notion of *design pattern* [20]. Defining a design pattern in his seminal work, Alexander, credited with invention of design patterns, states [1] that:

> Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem in such a way that you can use this solution a million times over without ever doing it the same way twice. *Christopher Alexander*

Generic types capture precisely this generality and the fingerprints define, in a planning-domain pattern language, the necessary components and relationships between them. The characterisations provided in figures 2 and 3 have a similar status to the design patterns of programs and can play a similar role: to support the construction of planning domain descriptions using well-understood structural components that capture common behaviours. Seen in this light, the automatic recognition of generic types is an attempt to recognise and capture design patterns in use within a planning domain. The exploitation of generic types as planning domain design patterns is at an early stage and we perceive there to be opportunities in domain engineering as well as in supporting efficient planning. Some early work illustrating the use of generic types in a domain engineering role can be found in [37].

The construction of planning domains in terms of generic types *ab initio* offers some important opportunities for reformulation. All of the techniques for abstraction of sub-problems can, of course, be used to reformulate problems in the ways already described, but in addition it is possible to use the existence of generic types to reformulate problems in canonical structures, or to add control information that is associated with the existence of generic types within a problem allowing automatic reformulation for planning systems that currently rely on hand-coded control rules [33].

## 5 Conclusions

Reformulation has played and continues to play a vital role in planning. In this paper we have concentrated on a particular strategy for the application of reformulation, based on the observation that planning problems typically contain sub-problems that have been tackled as research problems in their own right and for which efficient heuristic solvers have been developed that will inevitably out-perform generic planning technology. This approach is still at an early stage of development, but we have demonstrated that it is both possible and effective.

Planning is not alone in being faced with a wide variety of problems that are often composed of combinations of structured sub-problems. Reformulation in order to redeploy problem solving across multiple sub-solvers, each specialising in the solution of one kind of sub-problem, would appear to be a strategy that can be applied across a much

broader spectrum of combinatorial problem solving. Indeed, it seems likely that even the notion of a generic type is more general than to be applicable to planning problems alone, and its application to other areas of automatic reasoning could be a fruitful way in which to extend the power of reformulation across the whole field.

## References


1. C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel. *A Pattern Language*. Oxford University Press, 1977.
2. C.R. Anderson, D.E. Smith, and D.S. Weld. Conditional effects in Graphplan. In *Proc. of 4th International Conference on AI Planning Systems*, 1998.
3. F. Bacchus and F. Kabanza. Using temporal logic to express search control knowledge for planning. *Artificial Intelligence*, 116(1-2):123–191, 2000.
4. A. Blum and M. Furst. Fast Planning through Plan-graph Analysis. In *Proc. of 14th Internation Joint Conference on AI*, pages 1636–1642. Morgan Kaufmann, 1995.
5. B. Bonet and H. Geffner. Planning as heuristic search: new results. In *Proc. of 4th European Conference on Planning (ECP)*. Springer-Verlag, 1997.
6. A. Cimatti, M. Roveri, and P. Trverso. Strong planning in non-deterministic domains via model-checking. In *Proc. of 4th International Conference on AI Planning and Scheduling (AIPS'00)*, 2000.
7. M. Clark. Construction domains: a generic type solved. In *Proceedings of 20th Workshop of UK Planning and Scheduling Special Interest Group*, 2001.
8. S. Cresswell, M. Fox, and D. Long. Extending TIM domain analysis to handle ADL constructs. In L. McCluskey, editor, *Knowledge Engineering Tools and Techniques for AI Planning: AIPS'02 Workshop*, 2002.
9. M.B. Do and S. Kambhampati. Sapa: a domain-independent heuristic metric temporal planner. In *Proc. ECP-01*, 2001.
10. Minh Binh Do and S. Kambhampati. Solving planning graph by compiling it into a CSP. In *Proc. of 5th Conference on AI Planning Systems*, pages 82–91. AAAI Press, 2000.
11. B. Drabble and A. Tate. The use of optimistic and pessimistic resource profiles to inform search in an activity based planner. In *Proc. of 2nd Conference on AI Planning Systems (AIPS)*. AAAI Press, 1994.
12. S. Edelkamp. Mixed propositional and numeric planning in the model checking integrated planning system. In M. Fox and A. Coddington, editors, *Planning for Temporal Domains: AIPS'02 Workshop*, 2002.
13. M. Fox and D. Long. The automatic inference of state invariants in TIM. *Journal of AI Research*, 9:367–421, 1998.
14. M. Fox and D. Long. The detection and exploitation of symmetry in planning problems. In *Proc. of 16th Internation Joint Conference on AI*, pages 956–961. Morgan Kaufmann, 1999.
15. M. Fox and D. Long. Hybrid STAN: Identifying and Managing Combinatorial Sub-problems in Planning. In *Proc. of 17th International Joint Conference on AI*, pages 445–452. Morgan Kaufmann, 2001.
16. M. Fox and D. Long. Extending the exploitation of symmetries in planning. In *Proc. of 6th International Conference on AI Planning Systems (AIPS'02)*. AAAI Press, 2002.
17. M. Fox and D. Long. Fast temporal planning in a Graphplan framework. In M. Fox and A. Coddington, editors, *Planning for Temporal Domains: AIPS'02 Workshop*, 2002.
18. M. Fox, D. Long, S. Bradley, and J. McKinna. Using model checking for pre-planning analysis. In *AAAI Spring Symposium Series: Model-based Validation of Intelligence*. AAAI Press, 2001.

19. M. Fox, D. Long, and M. Hamdi. Handling multiple sub-problems within a planning domain. In *Proc. of 20th Workshop of UK Planning and Scheduling Special Interest Group*, 2001.
20. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of reusable software*. Addison-Wesley, 1995.
21. B.C. Gazen and C. Knoblock. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In *Proc. of 4th European Conference on Planning (ECP'97)*, 1997.
22. A. Gerevini and L. Schubert. Accelerating Partial Order Planners: Some Techniques for Effective Search Control and Pruning. *Journal of AI Research*, 5:95–137, 1996.
23. A. Gerevini and I. Serina. LPG: A planner based on local search for planning graphs. In *Proc. of 6th International Conference on AI Planning Systems (AIPS'02)*. AAAI Press, 2002.
24. J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of AI Research*, 14:253–302, 2000.
25. H. Kautz and B. Selman. Unifying SAT-based and graph-based planning. In *Proc. of 14th Internation Joint Conference on AI*, pages 318–325. Morgan Kaufmann, 1995.
26. J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an ADL subset. In *Proc. of 4th European Conference on Planning, Toulouse*, pages 273–285, 1997.
27. J. Kvarnstrom and P. Doherty. TALplanner: A temporal logic based forward chaining planner. *Annals of Mathematics and Artificial Intelligence*, 30(1-4):119–169, 2000.
28. P. Laborie and M. Ghallab. Planning with sharable resource constraints. In *Proc. of 14th International Joint Conference on AI*. Morgan Kaufmann, 1995.
29. D. Long and M. Fox. Automatic synthesis and use of generic types in planning. In *Proc. of 5th Conference on Artificial Intelligence Planning Systems (AIPS)*, pages 196–205. AAAI Press, 2000.
30. D. Long and M. Fox. Multi-processor scheduling problems in planning. In *Proc. of ICAI'01, Las Vegas*, 2001.
31. D. Long, M. Fox, L. Sebastia, and A. Coddington. An examination of resources in planning. In *Proc. of 19th UK Planning and Scheduling Workshop, Milton Keynes*, 2000.
32. D. Long and M.Fox. Planning with generic types. Technical report, Invited talk at IJCAI'01 (forthcoming Morgan-Kaufmann publication), 2001.
33. L. Murray. Reuse of control knowledge in planning domains. In L. McCluskey, editor, *Knowledge Engineering Tools and Techniques for AI Planning: AIPS'02 Workshop*, 2002.
34. D. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila. SHOP: Simple hierarchical orederd planner. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1999.
35. B. Nebel. On the compilability and expressive power of propositional planning formalisms. *Journal of AI Research*, 12:271–315, 2000.
36. E. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. of 1st International Conference on Principles of Knowledge Representation and Reasoning*, pages 324–332. San Francisco, CA, Morgan Kaufmann, 1989.
37. R. Simpson, L. McCluskey, D. Long, and M. Fox. Generic types as design patterns for planning domain specification. In L. McCluskey, editor, *Knowledge Engineering Tools and Techniques for AI Planning: AIPS'02 Workshop*, 2002.
38. B. Srivastava. RealPlan: Decoupling causal and resource reasoning in planning. In *Proc. of 17th National Conference on AI*, pages 812–818. AAAI/MIT Press, 2000.
39. P. van Beek and X. Chen. CPlan: A constraint programming approach to planning. In *Proc. of 16th National Conference on Artificial Intelligence*, pages 585–590. AAAI/MIT Press, 1999.

# Spatiotemporal Abstraction of Stochastic Sequential Processes

Sridhar Mahadevan

Department of Computer Science,
University of Massachusetts,
Amherst, MA 01003, USA
mahadeva@cs.umass.edu

**Abstract.** Probabilistic finite state machines have become a popular modeling tool for representing *sequential processes*, ranging from images and speech signals to text documents and spatial and genomic maps. In this paper, I describe two hierarchical abstraction mechanisms for simplifying the (estimation) learning and (control) optimization of complex Markov processes: *spatial* decomposition and *temporal* aggregation. I present several approaches to combining spatial and temporal abstraction, drawing upon recent work of my group as well as that of others. I show how spatiotemporal abstraction enables improved solutions to three difficult sequential estimation and decision problems: hidden state modeling and control, learning parallel plans, and coordinating with multiple agents.

## 1 Introduction

Abstraction has long been viewed as central to artificial intelligence (AI). A popular textbook defines abstraction as the "process of removing detail from a representation" [30]. Many approaches to abstraction have been pursued in the past several decades of research in AI. A common strategy is *constraint relaxation* where the problem is simplified by eliminating some conditions, as illustrated by logic-based planners such as ABSTRIPS [12] and by methods for discovering admissible heuristics [25]. This paper describes some recent work on *probabilistic abstraction* of stochastic sequential processes, which have become a common approach underlying many areas of AI.

Figure 1 characterizes a popular view of AI as the science underlying the design of *agents*: software or hardware artifacts that interact with an external environment through perception and action. What is unique about the agent-centered viewpoint is that it directs attention to the *sequential* interaction between an agent and its environment, and how to model the dynamics of such an interaction. Typically, the interaction is such that decisions (or observations) made earlier can impact later decisions.

Probabilistic finite state machines have become a popular paradigm for modeling sequential processes. In this representation, the interaction between an

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 33–50, 2002.

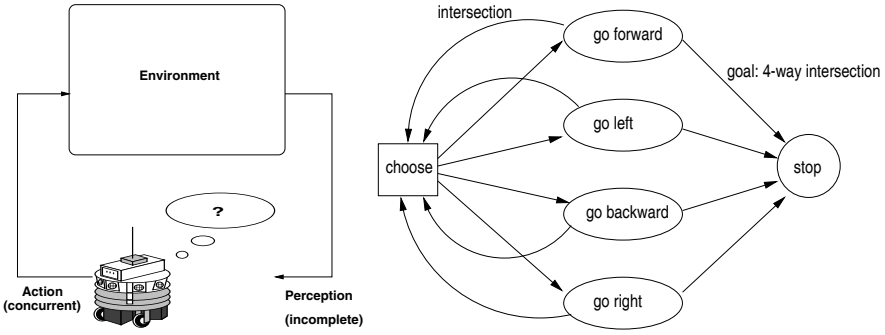



**Fig. 1.** The perception-action cycle of interaction between an agent and its environment can be modeled as a sequential process. A sequential program for a corridor navigation task represented as a finite state machine on the right. The set of observations generated as the agent executes this machine can be modeled as a Markov process.

agent and its environment is represented as a finite automata, whose *states* partition the past history of the interaction into equivalence classes, and whose *actions* cause (probabilistic) transitions between states. Here, state are a *sufficient statistic* for computing optimal (or best) actions, meaning past history leading to the state can be abstracted. This assumption is usually referred to as the *Markov* property.

Markov processes have become the mathematical foundation for much current work in reinforcement learning [33], decision-theoretic planning [1], information retrieval [7], speech recognition [10], active vision [20], and robot navigation [13]. In this paper, I focus on the abstraction of sequential Markov processes, and present two main strategies for "removing irrelevant detail": state aggregation/decomposition and temporal abstraction. State decomposition methods typically represent states as collections of *factored* variables [1], or simplify the automaton by eliminating "useless" states [3]. Temporal abstraction mechanisms, for example in hierarchical reinforcement learning [34,5,23], encapsulate lower-level observation or action sequences into a single unit at more abstract levels. For a unified algebraic treatment of abstraction of Markov decision processes that covers both spatial and temporal abstraction, the reader is referred to the paper by Ravi and Barto in these proceedings [27].

The main thesis of this paper is that combining spatial and temporal abstraction enables significant advances in solving difficult sequential estimation and decision problems. I focus on three specific problems – multiscale representations of hidden state, learning concurrent plans, and acquiring multiagent coordination strategies – and show how spatiotemporal abstraction is a powerful approach to solving instances of these well-known difficult problems.

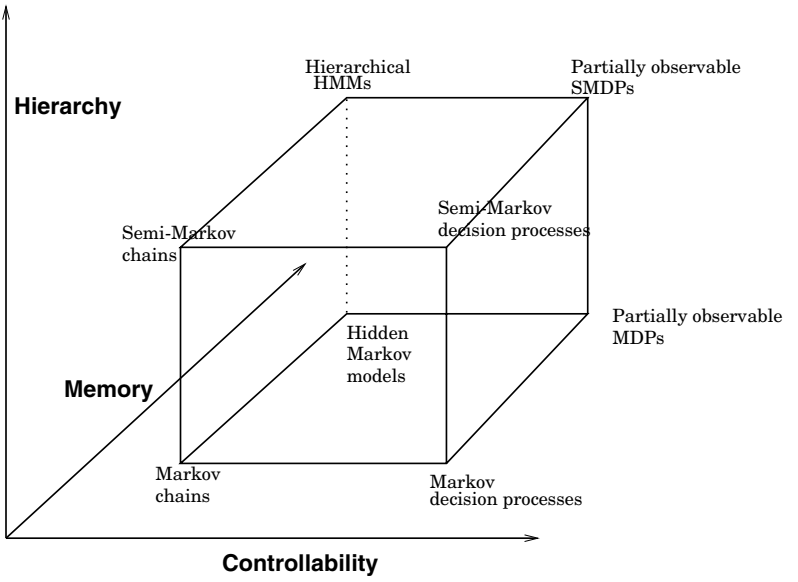



**Fig. 2.** A spectrum of Markov process models along several dimensions: whether agents have a choice of action, whether states are observable or hidden, and whether actions are unit-time (single-step) or time-varying (multi-step).

## 2 Markov Processes

Figure 2 illustrates eight Markov process models, arranged in a cube whose axes represent significant dimensions along which the models differ from each other. While a detailed description of each model is beyond the scope of this paper, I will provide examples of many of these models in the rest of this paper, beginning in this section with the basic MDP model.

A *Markov decision process* (MDP) [26] is specified by a set of states $S$, a set of allowable actions $A(s)$ in each state $s$, and a transition function specifying the next-state distribution $P^a_{ss'}$ for each action $a \in A(s)$. A reward or cost function $r(s, a)$ specifies the *expected* reward for carrying out action $a$ in state $s$. Solving a given MDP requires finding an optimal mapping or *policy* $\pi^* : S \to A$ that maximizes the long-term cumulative sum of rewards (usually discounted by some factor $\gamma < 1$) or the expected average-reward per step. A classic result is that for any MDP, there exists a stationary deterministic optimal policy, which can be found by solving a nonlinear set of equations, one for each state (such as by a successive approximation method called *value iteration*):

$$V^*(s) = \max_{a \in A(s)} \left( r(s, a) + \gamma \sum_{s'} P(s'|s, a) V^*(s') \right) \quad (1)$$

MDPs have been applied to many real-world domains, ranging from robotics [13, 16] to engineering optimization [2,17,17], and game playing [36]. In many such

domains, the model parameters (rewards, transition probabilities) are unknown, and need to be estimated from samples generated by the agent exploring the environment. Q-learning was a major advance in direct policy learning, since it obviates the need for model estimation [41]. Here, the Bellman optimality equation is reformulated using *action values* $Q^*(x, a)$, which represent the value of the non-stationary policy of doing action $a$ once, and thereafter acting optimally. Q-learning eventually finds the optimal policy asymptotically. However, much work is required in scaling Q-learning to large problems, and abstraction is one of the key components. Factored approaches to representing value functions are also emerging as a key approach to scaling to large problems [14].

## 3 Spatiotemporal Abstraction of Markov Processes

We now discuss strategies for hierarchical abstraction of Markov processes, including temporal abstraction, and spatial abstraction techniques.

### 3.1 Semi-Markov Decision Processes

Hierarchical decision-making models require the ability to represent lower-level policies over primitive actions as primitive actions at the next level (e.g., in the sequential machine in Figure 1, the "go forward" state might itself be comprised of a lower-level machine for moving through the corridor to the end, while avoiding obstacles). Policies over primitive actions are "semi-Markov" at the next level up, and cannot be simply treated as single-step actions over a coarser time scale over the same states.

Semi-Markov decision processes (SMDPs) have become the preferred language for modeling temporally extended actions. Unlike Markov decision processes (MDPs), the time between transitions may be several time units and can depend on the transition that is made. An SMDP is defined as a five tuple $(S,A,P,R,F)$, where $S$ is a finite set of states, $A$ is the set of actions, $P$ is the state and action transition probability function, $R$ is the reward function, and $F$ is a function giving probability of transition times for each state-action pair. The transitions are at decision epochs only. The SMDP represents snapshots of the system at decision points, whereas the so-called *natural process* [26] describes the evolution of the system over all times. Discrete-time SMDPs represent transition distributions as $F(s', N|s, a)$, which specifies the expected number of steps $N$ that action $a$ will take before terminating in state $s'$ starting in state $s$. For continuous-time SMDPs, $F(t|s, a)$ is the probability that the next decision epoch occurs within $t$ time units after the agent chooses action $a$ in state $s$ at a decision epoch. Q-learning generalizes nicely to discrete and continuous-time SMDPs. The Q-learning rule for discrete-time discounted SMDPs is

$$Q_{t+1}(s, a) \leftarrow Q_t(s, a)(1 - \beta) + \beta \left( R + \gamma^k \max_{a' \in A(s')} Q_t(s', a') \right)$$

where $\beta \in (0, 1)$, and action $a$ was initiated in state $s$, lasted for $k$ steps, and terminated in state $s'$, while generating a total discounted sum of rewards of $R$.

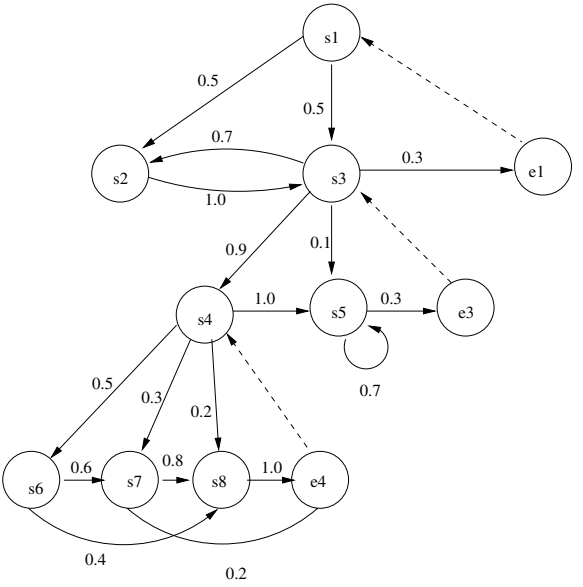



**Fig. 3.** An example hierarchical hidden Markov model. Only leaf nodes produce observations. Internal nodes can be viewed as generating sequences of observations.

Several frameworks for hierarchical reinforcement learning have been proposed, all of which are variants of SMDPs, including options [34], MAXQ [5], and HAMs [23]. We discuss some of these in more detail in the next section.

### 3.2 Hierarchical Hidden Markov Models

Hidden Markov models (HMMs) are a widely-used probabilistic model for representing time-series data, such as speech [10]. Unlike an MDP, states are not perceivable, and instead the agent receives an observation $o$ which can be viewed as being generated by a stochastic process $P(o|s)$. HMMs have been widely applied to many time-series problems, ranging from speech recognition [10], information extraction [7], and bioinformatics [11]. However, like MDPs, HMMs do not provide any direct way of representing higher-level structure that is often present in many practical problems. For example, an HMM can be used as a spatial representation of indoor enviroments [31], but typically such environments have higher order structures such as corridors or floors which are not made explicit in the underlying HMM model. As in the case with MDPs, in most practical problems, the parameters of the underlying HMM have to be learned from samples. The most popular method for learning an HMM model is the Baum-Welch procedure, which is itself a special case of the more general Expectation-Maximization (EM) statistical inference algorithm.

Recently, an elegant hierarchical extension of HMMs was proposed [6]. The HHMM generalizes the standard hidden Markov model by allowing hidden states

to represent stochastic processes themselves. An HHMM is visualized as a tree structure (see Figure 3) in which there are three types of states, production states (leaves of the tree) which emit observations, and internal states which are (unobservable) hidden states that represent entire stochastic processes. Each production state is associated with an observation vector which maintains distribution functions for each observation defined for the model. Each internal state is associated with a horizontal transition matrix, and a vertical transition vector. The horizontal transition matrix of an internal state defines the transition probabilities among its children. The vertical transition vectors define the probability of an internal state to activate any of its children. Each internal state is also associated with a child called an *end-state* which returns control to its parent. The end-states ($e1$ to $e4$ in Figure 3) do not produce observations and cannot be activated through a vertical transition from their parent.

Figure 3 shows a graphical representation of an example HHMM. The HHMM produces observations as follows:

1. If the current node is the root, then it chooses to activate one of its children according to the vertical transition vector from the root to its children.
2. If the child activated is a product state, it produces an observation according to an observation probability output vector. It then transitions to another state within the same level. If the state reached after the transition is the end-state, then control is returned to the parent of the end-state.
3. If the child is an abstract state then it chooses to activate one of its children. The abstract state waits until control is returned to it from its child end-state. Then it transitions to another state within the same level. If the resulting transition is to the end-state then control is returned to the parent of the abstract state.

The basic inference algorithm for hierarchical HMMs is a modification of the "inside-outside" algorithm for stochastic context-free grammars, and runs in $O(T^3)$ where $T$ is the length of the observation sequence. Recently, Murphy developed a much faster inference algorithm for hierarchical HMMs by mapping them onto a dynamic Bayes network [21].

### 3.3 Factorial Markov Processes

In many domains, states are comprised of collections of objects, each of which can be modeled as a multinomial or real-valued variable. For example, in driving, the state of the car might include the position of the accelerator and brake, the radio, the wheel angle etc. Here, we assume the agent-environment interaction can be modeled as a factored semi-Markov decision process, in which the state space is spanned by the Cartesian product of random variables $X = \{X_1, X_2, ..., X_n\}$, where each $X_i$ takes on values in some finite domain $Dom(X_i)$. Each action is either a primitive (single-step) action or a closed-loop policy over primitive actions.

Dynamic Bayes networks (DBNs) [4] are a popular tool for modeling transitions across factored MDPs. Let $X_i^t$ denote the state variable $X_i$ at time $t$

and $X_i^{t+1}$ the variable at time $t+1$. Also, let $A$ denote the set of underlying primitive actions. Then, for any set of actions represented by $\boldsymbol{a} \subseteq A$, the *Action Network* is specified as a two-layer directed acyclic graph whose nodes are $\{X_1^t, X_2^t, ..., X_n^t, X_1^{t+1}, X_2^{t+1}, ..., X_n^{t+1}\}$ and each node $X_i^{t+1}$ is associated with a *conditional probability table (CPT)* $P(X_i^{t+1}|\phi(X_i^{t+1}), \boldsymbol{a})$ in which $\phi(X_i^{t+1})$ denotes the parents of $X_i^{t+1}$ in the graph. The transition probability $P(X^{t+1}|X^t, \boldsymbol{a})$ is then defined by: $P(X^{t+1}|X^t, \boldsymbol{a}) = \prod_i^n P(X_i^{t+1}|w_i, \boldsymbol{a})$ where $w_i$ is a vector whose elements are the values of the $X_j^t \in \phi(X_i^{t+1})$.

Figure 4 shows a popular toy problem called the Taxi Problem [5]) in which a taxi inhabits a 7-by-7 grid world. This is an episodic problem in which the taxi (with maximum fuel capacity of 18 units) is placed at the beginning of each episode in a randomly selected location with a randomly selected amount of fuel (ranging from 8 to 15 units). A passenger arrives randomly in one of the four locations marked as R(ed), G(reen), B(lue), and Y(ellow) and will select a random destination from these four states to be transported to. The taxi must go to the location of the passenger (the "source"), pick up the passenger, move to the destination location (the "destination) and put down the passenger there. The episode ends when either the passenger is transported to the desired destination, or the taxi runs out of fuel. Treating each of taxi position, passenger location, destination and fuel level as state variables, we can represent this problem as a factorial MDP with four state variables each taking on values as explained above. Figure 4 shows a factorial representation of taxi domain for *Pickup* and *Fillup* actions.

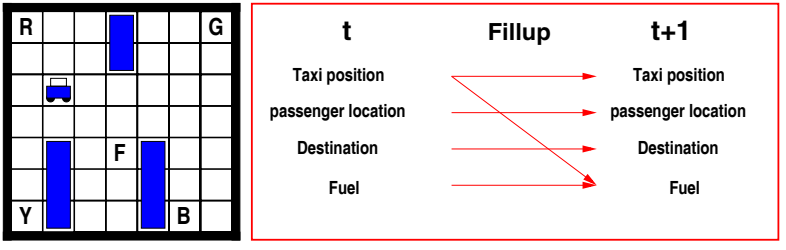



**Fig. 4.** The taxi domain is an instance of a factored Markov process, where actions such as fillup can be represented compactly using dynamic Bayes networks.

Other examples include the *mixed memory factorial Markov model* and its extension to factorial MDPs [29]. In factorial MDPs, the transition model is additionally decomposed into transition sub-components, each describing a transition sub-model for a particular state variable given the current instantiation of the set of state variables. Mixed memory representation of the transition probabilities models each sub-component as a mixture of simpler dynamical models, each describing a cross-correlation between a pair of state variables. The parameters of this model can be fitted iteratively using an EM procedure.

### 3.4 Structural Decomposition of Markov Processes

Other related techniques for decomposition of large MDPs have been explored, and some of these are illustrated in Figure 5. A simple decomposition strategy is to split a large MDP into sub-MDPs, which interact "weakly" [23,34,3]. An example of weak interaction is navigation, where the only interaction among sub-MDPs is the states that connect different rooms together. Another strategy is to decompose a large MDP using the set of available actions, such as in air campaign planning problem [19], or in conversational robotics [24]. An even more intriguing decomposition strategy is when sub-MDPs interact with each other through shared parameters. The transfer line optimization problem from manufacturing is a good example of such a parametric decomposition [40].

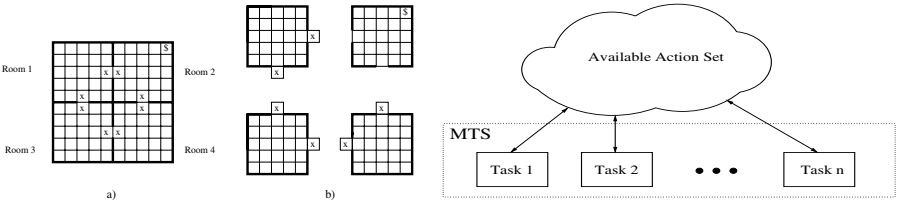



**Fig. 5.** State and action-based decomposition of Markov processes.

## 4 Spatiotemporal Abstraction: Three Case Studies

This section describes some recent research from my group on exploiting spatiotemporal abstraction to produce improved solutions to three difficult problems: learning concurrent actions and multiagent coordination, and using memory to deal with hidden state.

### 4.1 Learning Concurrent Plans

I now describe a probabilistic model for learning concurrent plans over temporally extended actions. Figure 6 illustrates a toy example of concurrent planning. The general problem is as follows. The agent is given a set of temporally extended actions, each of which can be viewed as a (fixed or previously learned) "subroutine" for choosing actions over a subspace of the overall state space. The goal of the agent is to learn to construct a closed-loop plan (or policy) that allows multiple concurrent subroutines to be executed in parallel (and in sequence) to achieve the task at hand. For multiple actions to be executed concurrently, their joint semantics must be well-defined. Concurrency is facilitated by assuming states are not atomic, but structured as a collection of (discrete or continuous) variables, and the effect of actions on such sets of variables can be captured by a compact representation, such as a dynamic Bayes net (DBN) [4].

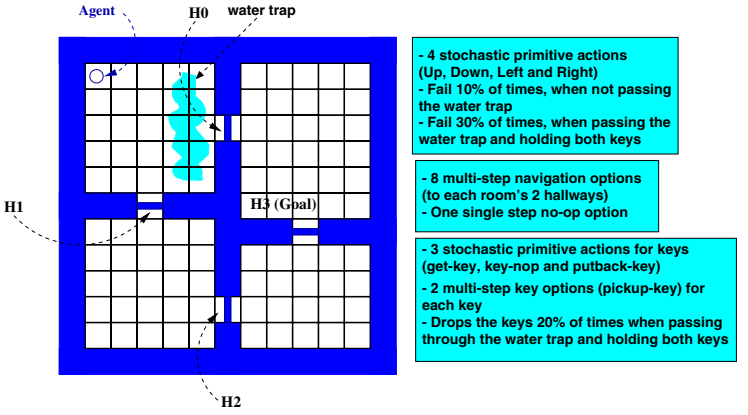



**Fig. 6.** A grid world problem to illustrate concurrent planning: the agent is given subroutines for getting to each door from any interior room state, and for opening a locked door. It has to learn the shortest path to the goal by concurrently combining these subroutines. The agent can reach the goal more quickly if it learns to parallelize the subroutine for retrieving the key before it reaches a locked door. However, retrieving the key too early is counterproductive since it can drop with some probability.

Since multiple concurrent actions may not terminate synchronously, the notion of a decision epoch needs to be generalized. For example, a decision epoch can occur when any one of the actions currently running terminates. We refer to this as the $T_{any}$ termination condition. Alternatively, a decision epoch can be defined to occur when all actions currently running terminate, which we refer to as the $T_{all}$ condition.

For concreteness, we will describe the concurrent planning framework using the *options* formalism [34]. The treatment here is restricted to options over discrete-time SMDPs and deterministic policies, but the main ideas extend readily to other hierarchical formalisms [5,23] and to continuous-time SMDPs [8]. More formally, an option $o$ consists of three components: a policy $\pi : S \to A$, a termination condition $\beta : S \to [0, 1]$, and an initiation set $I \subseteq S$, where $I$ denotes the set of states $s$ in which the option can be initiated. For any state $s$, if option $\pi$ is taken, then primitive actions are selected based on $\pi$ until it terminates according to $\beta$. An option $o$ is a *Markov option* if its policy, initiation set and termination condition depend stochastically only on the current state $s \in S$. An option $o$ is *semi-Markov* if its policy, initiation set and termination condition are dependent on all prior history since the option was initiated. For example, the option *exit-room* in the grid world environment shown in Figure 6, in which states are the different locations in the room, is a Markov option, since for a given location, the direction to move to get to the door can be computed given the current state.

A hierarchical policy over subroutines or options can be defined as follows. The Markov policy over options $\mu : S \rightarrow O$ (where $O$ is the set of all options) selects an option $o \in O$ at time $t$ using the function $\mu(s_t)$. The option $o$ is then initiated in $s_t$ until it terminates at a random time $t + k$ in some state $s_{t+k}$ according to the termination condition, and the process repeats in $s_{t+k}$.

The multistep state transition dynamics over options is defined using the discount factor to weight the probability of transitioning. Let $p^o(s, s', k)$ denote the probability that the option $o$ is initiated in state $s$ and terminates in state $s'$ after $k$ steps. Then $p(s'|s, o) = \sum_{k=1}^{\infty} p^o(s, s', k)\gamma^k$. If multi-step models of options and rewards are known, optimal hierarchical plans can be found by solving a generalized Bellman equation over options similar to Equation 1.

The sequential subroutine (option) model is now generalized to *concurrent multi-options*. Initially, assume the set of options can be partitioned into mutually exclusive *coherent* subsets, where options from disjoint subsets can always be parallelized since they affect different state variables. For example, turning the radio off and pressing the brake can always be executed in parallel since they affect different state variables.

We now define multi-options (denoted below by $\boldsymbol{o}$) more precisely. Let $o \equiv < I, \pi, \beta >$ be a standard option with state space $S_o$ governed by the set of state variables $W_o = \{w_1^o, w_2^o, ..., w_{n_o}^o\}$. Let $\varphi_o \subset W_o$ denote the subset of state variables that evolve by some other processes (e.g. other options) and independent of $o$, and let $\Omega_o = W_o - \varphi_o$ denote the subset of state variables that evolve solely based on the option $o$. We refer to the class of options with this property as *partially-factored* options. Two options $o_i$ and $o_j$ are called *coherent* if (1) they are both *partially-factored* options, and (2) $\Omega_{o_i} \cap \Omega_{o_j} = \emptyset$ (this condition ensures these two options will not affect the same portion of the state space so that they can safely run in parallel). In the driving example, *turn radio on* and *brake* options are coherent, but *turn-right* and *accelerate* options are not coherent, since the state variable *position* is controlled by both these actions.

Now, assume $O$ is the set of available options and $\{C_1, C_2, ..., C_n\}$ are $n$ classes of options that partition $O$ into disjoint classes such that any two options belonging to different classes are coherent (can run in parallel), and any two options within the same class are not coherent. Clearly any set of options generated by drawing each option from a separate class can safely be run in parallel. Given the above definitions, we can define the multi-option model as a 4-tuple $(S, \boldsymbol{O}, P, r)$, where $S$ is the state space spanned by the cartesian product set over state variables, $\boldsymbol{O}$ is the set of all possible concurrent multi-options $\subseteq C_1 \times C_2 \times ... \times C_n$, $P$ is the transition probability specifying the state dynamics under any multi-option, and $r : S \times \boldsymbol{O} \rightarrow \mathcal{R}$ is the expected reward for taking a multi-option $\boldsymbol{o} \in \boldsymbol{O}$.

When multi-option $\boldsymbol{o}$ is executed in state $s$, a set of $m$ options $o_i \in \boldsymbol{o}$ are initiated. Each option $o_i$ will terminate at some random time $t_{o_i}$. We can define the event of *termination* for a multi-option based on either of the following events: (1) $T_{all} = \max_i(t_{o_i})$: when all the options $o_i \in \boldsymbol{o}$ terminate according to $\beta_i(s)$, multi-option $\boldsymbol{o}$ is declared terminated (2) $T_{any} = \min_i(t_{o_i})$: when any (i.e., the

first) of the options terminate, the rest of the options that are not terminated at that point in time are interrupted. Under either definition of termination, the following result holds.

**Theorem 1.** *Given a Markov decision process, and a set of concurrent Markov options defined on it, the decision process that selects only among multi-options, and executes each one until its termination according to the either $T_{all}$ or $T_{any}$ termination condition forms a semi-Markov decision process.*

The proof requires showing that the state transition dynamics $p(s', N \mid \boldsymbol{o}, s)$ and the rewards $r(s, \boldsymbol{o})$ over any concurrent option $\boldsymbol{o}$ defines a semi-Markov decision process [28]. The significance of this result is that SMDP Q-learning methods can be extended to learn to concurrent plans under this model. The extended SMDP Q-learning algorithm for learning to plan with concurrent options updates the multi-option-value function $Q(s, \boldsymbol{o})$ after each decision epoch where the multi-option $\boldsymbol{o}$ is taken in some state $s$ and terminates in $s'$ (under either termination condition):

$$Q(s, \boldsymbol{o}) \leftarrow Q(s, \boldsymbol{o})(1 - \beta) + \beta \left[ R + \gamma^k \max_{\boldsymbol{o'} \in O_{s'}} Q(s', \boldsymbol{o'}) \right] \tag{2}$$

where $k$ denotes the number of time steps between initiation of the multi-option $\boldsymbol{o}$ in state $s$ and its termination in state $s'$, and $R$ denotes the cumulative discounted reward over this period. The result of using this algorithm on the simple grid world problem in shown in Figure 7. The figure illustrates the difference in performance under different termination conditions ($T_{all}$, $T_{any}$, and $T_{cont}$).

The concurrent option model can be extended to allow cases when options executing in parallel modify the same shared variables at the same time. Using a DBN representation of concurrent actions, the above theorem continues to hold in this case (and the concurrent SMDP Q-learning method works as well).

### 4.2 Learning Multiagent Task-Level Coordination Strategies

The second case study uses hierarchical abstraction to learn multiagent coordination strategies. Figure 8 illustrates a robot trash collection task, where the two agents $A1$ and $A2$ will maximize their performance at the task if they learn to coordinate with each other. Here, we want to design learning algorithms for *cooperative* multiagent tasks [42], where the agents learn the coordination skills by trial and error. The key idea here is that coordination skills are learned more efficiently if agents learn to synchronize using a hierarchical representation of the task structure [32]. In particular, rather than each robot learning its response to low-level primitive actions of the other robots (for instance, if A1 goes forward, what should A2 do), they learn high-level coordination knowledge (what is the utility of A2 picking up trash from T1 if A1 is also picking up from the same bin, and so on). The proposed approach differs significantly from previous work in multiagent reinforcement learning [15,35] in using hierarchical task structure

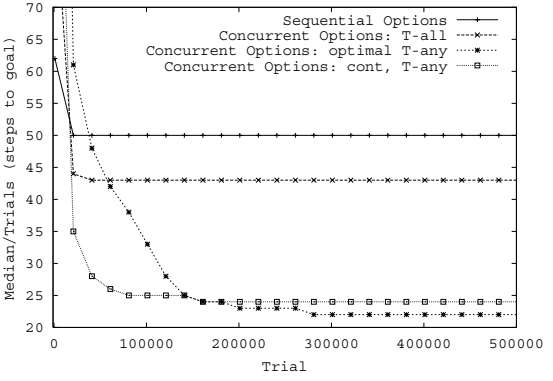



**Fig. 7.** This graph compares an SMDP technique for learning concurrent plans (under various termination conditions) with a slower "get-to-door-then-pickup-key" sequential plan learner. The concurrent learners outperform the sequential learner, but the choice of termination affects the speed and quality of the final plan.

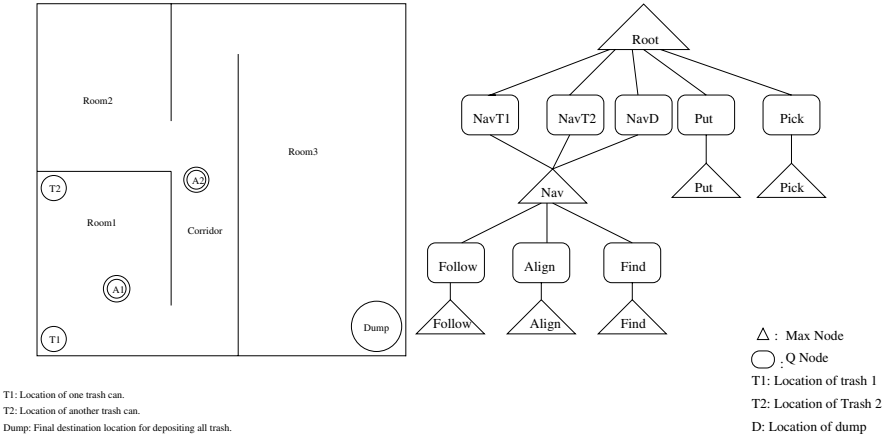



**Fig. 8.** A two-robot ($A1$ and $A2$) trash collection task. The robots can learn to coordinate much more rapidly using the task structure than if they attempted to coordinate at the level of primitive movements.

to accelerate learning, and as well in its use of concurrent temporally extended actions.

One general approach to learning task-level coordination is to extend the above concurrency model to the joint state action space, where base level policies remain fixed. However, an extension of this approach is now presented, where agents learn coordination skills and the base-level policies all at once. However, convergence to optimal (hierarchical) policies is no longer assured since lower

level (subroutine) policies are varying at the same time when learning higher level routines. The ideas extend to other formalisms also, but for the sake of clarity, we focus on the MAXQ value function decomposition approach [5]. This decomposition is based on storing the value function in a distributed manner across all nodes in a task graph. The value function is computed on demand by querying lower level (subtask) nodes whenever a high level (task) node needs to be evaluated.

It is necessary to generalize the MAXQ decomposition from its original sequential single-agent setting to the concurrent multiagent coordination problem. Let $\boldsymbol{o} = (\boldsymbol{o_1}, ..., \boldsymbol{o_n})$ denote a concurrent action, where $\boldsymbol{o_i}$ is the set of concurrent subprocesses under the control of agent $i$. Let $\boldsymbol{s} = (s_1, ..., s_n)$ denote the joint state. The joint action value function $Q(p, \boldsymbol{s}, \boldsymbol{o})$ represents the value of concurrent action $\boldsymbol{o}$ in joint state $\boldsymbol{s}$, in the *context* of doing parent task $p$.

The MAXQ decomposition of the Q-function relies on a key principle: the reward function for the parent task is essentially the value function of the child subtask. This principle can be extended to joint concurrent action values as shown below. Define the *completion function* for agent $j$ as $C^j(p, s_j, \boldsymbol{o})$ as the expected cumulative discounted (or average-adjusted) reward of agent $j$ completing concurrent subtask $\boldsymbol{o_j}$ in the context of doing parent task $p$, when the other agents are performing concurrent subtasks $o_k, \forall k \in \{1, ..., n\}, k \not= j$. The joint concurrent action value function $Q(p, \boldsymbol{s}, \boldsymbol{o})$ is now approximated by each agent $j$ (given only its local state $s_j$) as:

$$Q^j(p, s_j, \boldsymbol{o}) \approx V^j(\boldsymbol{o_j}, s_j) + C^j(p, s_j, \boldsymbol{o}) \tag{3}$$

$$V^j(p, s_j) = \begin{cases} \max_{\boldsymbol{o_k}} Q^j(p, s_j, \boldsymbol{o_k}) & \text{if parent task p is composite} \\ \sum_{s'_j} P(s'_j \mid s_j, p) R(s'_j \mid s_j, p) & \text{if p is a primitive action} \end{cases}$$

The first term in the $Q(p, s_j, \boldsymbol{o})$ expansion above refers to the discounted sum of rewards received by agent $j$ for doing concurrent action $\boldsymbol{o_j}$ in state $s_j$. The second term "completes" the sum by accounting for rewards earned for completing the parent task $p$ after finishing $\boldsymbol{o_j}$. The completion function is updated from sample values using an SMDP learning rule. Note that the correct action value is approximated by only considering local state $s_j$ and also by ignoring the effect of concurrent actions $\boldsymbol{o_k}, k \not= j$ by other agents when agent $j$ is performing $\boldsymbol{o_j}$. In practice, a human designer can configure the task graph to store joint concurrent action values at the highest (or lower than the highest as needed) level(s) of the hierarchy as needed.

To illustrate the use of this decomposition in learning multiagent coordination, for the two-robot trash collection task, if the joint action-values are restricted to only the highest level of the task graph under the root, we get the following value function decomposition for agent $A1$:

$$Q^1(Root, s_1, (NavT1, NavT2)) \approx V_t^1((NavT1), s_1) + C_t^1(Root, s_1, (NavT1, NavT2))$$

which represents the value of agent $A1$ doing task $NavT1$ in the context of the overall $Root$ task, when agent $A2$ is doing task $NavT2$. Note that this value is decomposed into the value of agent $A1$ doing $NavT1$ subtask itself and the

completion sum of the remainder of the overall task done by both agents. In this example, the multiagent MAXQ decomposition embodies the constraint that the value of $A1$ navigating to trash bin $T1$ is independent of whatever $A2$ is doing.

### 4.3 Hierarchical Memory

When agents learn to act concurrently in real-world environments, the true state of the environment is usually hidden. To address this issue, we need to combine the above methods for learning concurrency and coordination with methods for estimating (joint) hidden state and actions. We have explored two multiscale memory models [9,39]. *Hierarchical Suffix Memory* (HSM) [9] generalizes the suffix tree model [18] to SMDP-based temporally extended actions. Suffix memory constructs state estimators from finite chains of observation-action-reward triples. In addition to extending suffix models to SMDP actions, HSM also uses multiple layers of temporal abstraction to form longer-term memories at more abstract levels. Figure 9 illustrates this idea for robot navigation for the simpler case of a linear chain, although the tree-based model has also been investigated. An important side-effect is that the agent can look back many steps back in time while ignoring the exact sequence of low-level observations and actions that transpired. Tests in a robot navigation domain showed that HSM outperformed "flat" suffix tree methods, as well as hierarchical methods that used no memory [9]. POMDPs are theoretically more powerful than finite memory models,

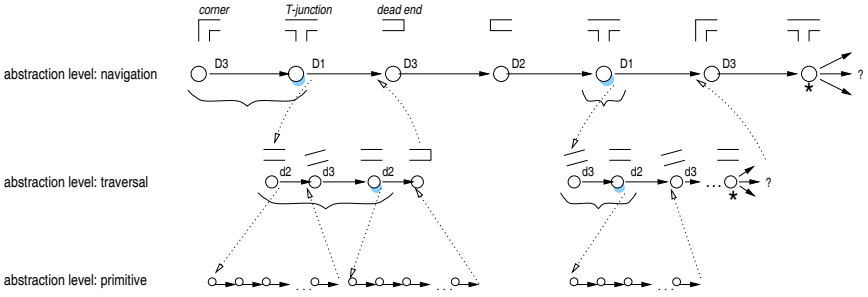



**Fig. 9.** A hierarchical suffix memory state estimator for a robot navigation task. At the abstract (navigation) level, observations and decisions occur at intersections. At the lower (corridor-traversal) level, observations and decisions occur within the corridor. At each level, each agent constructs states out of its past experience with similar history (shown with shadows).

but past work on POMDPs has mostly studied "flat" models for which learning and planning algorithms scale poorly with model size. We have developed a new *hierarchical POMDP* framework termed H-POMDPs (see Figure 10) [39], by extending the hierarchical hidden Markov model (HHMM) [6] to include rewards and (temporally extended) actions. We have developed a hierarchical EM

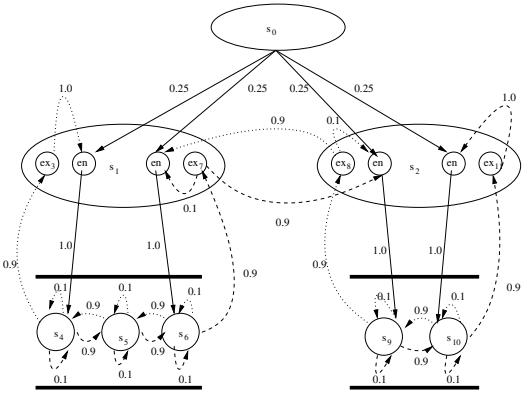



**Fig. 10.** An example hierarchical POMDP model representing two adjacent corridors in a robot navigation task. The model has two primitive actions, "go-left" indicated with the dotted arrows and "go-right" indicated with the dashed arrows. This HPOMDP has two (unobservable) abstract states $s_1$ and $s_2$ and each abstract state has two entry and two exit states. The (hidden) product state $s_4$, $s_5$, $s_6$, $s_9$, and $s_{10}$ have associated observation models.

algorithm for learning the parameters of an H-POMDP model from sequences of observations and actions. Extensive tests on a robot navigation domain show learning and planning performance is much improved over flat POMDP models [39,38]. The hierarchical EM-based parameter estimation algorithm scales more gracefully to large models because previously learned sub-models can be reused when learning higher levels. Also, the effect of temporally extended actions in H-POMDPs (e.g. exit the corridor) can be modeled at abstract and product level states, which supports planning at multiple levels of abstraction. H-POMDPs have an inherent advantage in allowing belief states to be computed at different levels of the tree. In addition, there is often less uncertainty at higher levels (e.g., a robot is more sure of which corridor it is in, rather than exactly which low level state). A number of heuristics for mapping belief states to action provide good performance in robot navigation (e.g, the most-likely-state (MLS) heuristic assumes the agent is in the state corresponding to the "peak" of the belief state distribution) [13,31,22]. Such heuristics work much better in H-POMDPs because they can be applied at multiple levels, and belief states over abstract states usually have lower entropy (see Figure 11). For a detailed study of the H-POMDP model, as well as its application to robot navigation, see [37].

## 5 Conclusions

In this paper, I described some general approaches to solving large sequential process models through spatiotemporal abstraction. In particular, I presented a framework for learning parallel plans that combined factored state representations with temporally extended actions. In addition, I described how this concur-

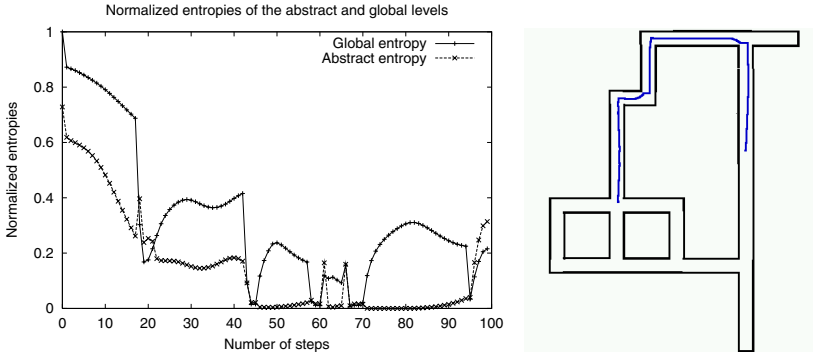



**Fig. 11.** This plot shows a sample robot navigation run whose trace is on the right, where positional uncertainty (measured by belief state entropy) at the abstract (corridor) level is less than at the product state level. Spatiotemporal abstraction reduces the uncertainty and requires less frequent decision-making, allowing the robot to get to goals without inital positional information.

rency framework could be extended to learn multiagent coordination strategies by using the overall task hierarchy. Finally, I showed how abstraction can help alleviate the problem of hidden state by building multiresolution state estimators. These case studies are some inital steps towards a more unified and rigorous approach to abstraction of sequential processes.

**Acknowledgements.** I am indebted to my current and former graduate students whose work is summarized in this paper, including Natalia Hernandez-Gardiol, Mohammad Ghavamzadeh, Rajbala Makar, Silviu Minut, Khashayar Rohanimanesh, and Georgios Theocharous. This research was supported in part by grants from the National Science Foundation (KDI), the Defense Advanced Research Projects Agency (MARS and DR programs), Michigan State University, and the University of Massachusetts, Amherst.

## References

1. C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1–2):49–107, 2000.
2. R.H. Crites and A.G. Barto. Elevator group control using multiple reinforcement learning agents. *Machine Learning*, 33:235–262, 1998.
3. T. Dean and R. Givan. Model minimization in markov decision processes. *Proceedings of AAAI*, 1997.
4. T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.
5. T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *International Journal of Artificial Intelligence Research*, 13:227–303, 2000.

6. S. Fine, Y. Singer, and N. Tishby. The Hierarchical Hidden Markov Model: Analysis and Applications. *Machine Learning*, 32(1), July 1998.
7. Dayne Freitag and Andrew Kachites McCallum. Information extraction with hmms and shrinkage. In *Proceedings of the AAAI-99 Workshop on Machine Learning for Informatino Extraction*, 1999.
8. M. Ghavamzadeh and S. Mahadevan. Continuous-time hierarchical reinforcement learning. In *Proceedings of the Eighteenth International Conference on Machine Learning*, 2001.
9. N. Hernandez and S. Mahadevan. Hierarchical memory-based reinforcement learning. *Proceedings of Neural Information Processing Systems*, 2001.
10. F. Jellinek. *Statistical Methods in Speech Recognition.* MIT Press, 2000.
11. K. Karplus, C. Barrett, and R. Hughey. Hidden markov models for detecting remote protein homologies, 1998.
12. Craig A. Knoblock. An analysis of ABSTRIPS. In James Hendler, editor, *Artificial Intelligence Planning Systems: Proceedings of the First International Conference (AIPS 92)*, pages 126–135, College Park, Maryland, USA, 1992. Morgan Kaufmann.
13. S. Koenig and R. Simmons. Xavier: A robot navigation architecture based on partially observable markov decision process models. In D. Kortenkamp, P. Bonasso, and Murphy. R., editors, *AI-based Mobile Robots: Case-studies of Successful Robot Systems*. MIT Press, 1997.
14. Daphne Koller and Ronald Parr. Computing factored value functions for policies in structured mdps. *16th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1332–1339, 1999.
15. M. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163, 1994.
16. S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55:311–365, 1992. Appeared originally as IBM TR RC16359, Dec 1990.
17. S. Mahadevan, N. Marchalleck, T. Das, and A. Gosavi. Self-improving factory simulation using continuous-time average-reward reinforcement learning. In *Proc. 14th International Conference on Machine Learning*, pages 202–210. Morgan Kaufmann, 1997.
18. Andrew K. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, 1995.
19. N. Meuleau, M. Hauskrecht, K. Kim, L. Peshkin, L. Kaelbling, T. Dean, and C. Boutilier. Solving very large weakly coupled markov decision processes. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 1998.
20. S. Minut and S. Mahadevan. A reinforcement learning model of selective visual attention. In *Fifth International Conference on Autonomous Agents*, 2001.
21. K. Murphy and M. Paskin. Linear time inference in hierarchical hmms. *Proceedings of Neural Information Processing Systems*, 2001.
22. I. Nourbakhsh, R. Powers, and S. Birchfield. Dervish: An office-navigation robot. *AI Magazine*, 16(2):53–60, 1995.
23. R.E. Parr. *Hierarchical Control and Learning for Markov Decision Processes*. PhD Thesis, University of California, Berkeley, 1998.
24. J. Pineau, N. Roy, and S. Thrun. A hierarchical approach to POMDP planning and execution. In *Workshop on Hierarchy and Memory in Reinforcement Learning (ICML 2001)*, Williams College, MA, June 2001.

25. A. Prieditis. Machine discovery of admissible heuristics, 1993.
26. M. L. Puterman. *Markov Decision Processes*. Wiley Interscience, New York, USA, 1994.
27. B. Ravi and A. Barto. Model minimization in hierarchical reinforcement learning. In *Symposium on Abstraction and Reformulation (SARA 2002)*. Springer Verlag, 2002.
28. K. Rohanimanesh and S. Mahadevan. Decision-theoretic planning with concurrent temporally extended actions. In *17th Conference on Uncertainty in Artificial Intelligence*, 2001.
29. K. Rohanimanesh and S. Mahadevan. Incremental learning of factorial markov decision processes. under preparation, 2002.
30. S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 1994.
31. Hagit Shatkay and Leslie Pack Kaelbling. Learning topological maps with weak local odometric information. In *IJCAI (2)*, pages 920–929, 1997.
32. T. Sugawara and V. Lesser. Learning to improve coordinated actions in cooperative distributed problem-solving environments. *Machine Learning*, 33:129–154, 1998.
33. R. Sutton and A. Barto. *An introduction to reinforcement learning*. MIT Press, Cambridge, MA., 1998.
34. R. Sutton, D. Precup, and S. Singh. Between MDPs and Semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.
35. M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337, 1993.
36. Gerald Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–278, 1992.
37. G. Theocharous. *Hierarchical Learning and Planning in Partially Observable Markov Decision Processes*. PhD Thesis, Michigan State University, 2002.
38. G. Theocharous and S. Mahadevan. Approximate planning with hierarchical partially observable markov decision processs for robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2002.
39. G. Theocharous, K. Rohanimanesh, and S. Mahadevan. Learning hierarchical partially observable markov decision processs for robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2001.
40. G. Wang and S. Mahadevan. Hierarchical optimization of policy-coupled semi-Markov decision processes. In *Proc. 16th International Conf. on Machine Learning*, pages 464–473. Morgan Kaufmann, San Francisco, CA, 1999.
41. C. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, England, 1989.
42. G. Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA., 1999.

# State Space Relaxation and Search Strategies in Dynamic Programming

Aristide Mingozzi

Department of Mathematics, University of Bologna, Bologna, Italy
mingozzi@csr.unibo.it

**Abstract.** Few combinatorial optimization problems can be solved effectively by dynamic programming as the number of the vertices of the state space graph can be enormous.
State space relaxation, introduced by Christofides, Mingozzi and Toth [1] is a general relaxation method whereby the state-space graph of a given dynamic programming recursion is relaxed in such a way that the solution of the associated recursion in the relaxed state-space provides a lower bound to the original problem. This talk gives a survey of this methodology and gives, as examples, applications to some combinatorial optimization problems including the traveling salesman problem (TSP). We describe a more general relaxation method for dynamic programming which produces a "reduced state space" containing a subset of the original states and the state space relaxation of the other states. Subgradient optimization and "state space decompression" are discussed as methods for improving the resulting lower bounds. We describe an iterative optimal search strategy in the original state space using bounding functions based on the reduced state space which explores, at each iteration, only a limited subset of states of the original state space graph. This procedure can also be used as a heuristic simply by limiting the maximum number of iterations. We give, as examples, applications to the TSP with time windows and precedence constraints and to the shortest path problem with additional constraints.

## References

1. Christofides, N., Mingozzi, A., Toth, P.: State-space relaxation procedures for the computation of bounds to routing problems. Networks **11** (1981)
2. Baldacci, R.: Algorithms for Location and Routing Problems in Distribution Systems. PhD thesis, Imperial College, London (1999)
3. Bodin, L., Mingozzi, A., Baldacci, R., Ball, M.: The rollon-rolloff vehicle routing problem. Transportation Science **34** (2000)
4. Bianco, L., Mingozzi, A., Ricciardelli, S., Spadoni, M.: Exact and heuristic prodedures for the tsp with precedence constraints. Infor **32** (1994)
5. Christofides, N., Mingozzi, A., Toth, P.: Exact algorithms for the tsp with additional constraints. Technical Report IC, OR 80, Imperial College (1979)
6. Mingozzi, A., Bianco, L., Ricciardelli, S.: Dynamic programming strategies for the tsp with time windows and precedence constraints. Operations Research **45** (1997)

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, p. 51, 2002.

# Admissible Moves in Two-Player Games

Tristan Cazenave

Labo IA, Université Paris 8
2 rue de la Liberté, 93526, St-Denis, France
cazenave@ai.univ-paris8.fr

**Abstract.** Some games have abstract properties that can be used to design admissible heuristics on moves. These admissible heuristics are useful to speed up search. They work well with depth-bounded search algorithms such as Gradual Abstract Proof Search that select moves based on the distance to the goal. We analyze the benefits of these admissible heuristics on moves for rules generation and search. We give experimental results that support our claim for the game of AtariGo.

## 1 Introduction

In some games, abstract properties can be used to design admissible heuristics on the minimal number of moves required to win the game. It is possible to relax the rules of a game and play admissible moves in the game with relaxed rules. The number of moves to win is always lower in the relaxed game than in the real game. The interest of relaxation is that the minimal number of moves can be computed faster than in the original game. The abstract knowledge on the moves can be used to select a subset of relevant threat moves when using a threat based search algorithm. It can also be used to stop a depth-bounded search when the number of admissible moves required to win is greater than twice the depth of the search.

In the paper, the attacker is the player who tries to win a game, and the defender is the player who tries to prevent the attacker from winning. For example, if the game is to make a group live in the game of Go, the attacker is the player who plays moves to live, and the defender is the player who tries to kill the group. When the game is to connect two strings in the game of Go or in the game of Hex, the attacker tries to connect the strings and the defender tries to prevent the attacker from doing so.

We present examples of admissible heuristics on moves, as well as experimental results and some methods relevant to the generation and the use of admissible moves in two-players games. Where experimental results are available, we mention them. Whenever it is possible, we also outline ideas that are currently under investigation in the hope to stimulate research on admissible heuristics in games. In the second section, the game of AtariGo is described. In the third section, we explain how admissible heuristics can be designed in two-player games. In the fourth section, we relate the admissible heuristics on the number of moves that have to be played in order to win a game to threat based search algorithms. We

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 52–63, 2002.

also describe Gradual Abstract Proof Search and we give experimental results quantifying the usefulness of abstract knowledge for the game of AtariGo solved by the Abstract Gradual Proof Search algorithm. In the fifth section, we outline the interest of admissible heuristics for the automatic generation of rules. The sixth section details the automatic generation of admissible heuristics on moves by transforming a logic program of the rules of the game. The last section outlines future work and concludes.

## 2 AtariGo

AtariGo is used to teach beginners to play the game of Go. The board used to play AtariGo is a square grid. It can be played on any board size. It is usually played on a small board so that games do not take too much time. Teachers also often choose to start with a crosscut in the centre of the board in order to have an unstable position. We have tested the usefulness of different abstraction levels on the game starting with a crosscut in the centre of a 6x6 board.

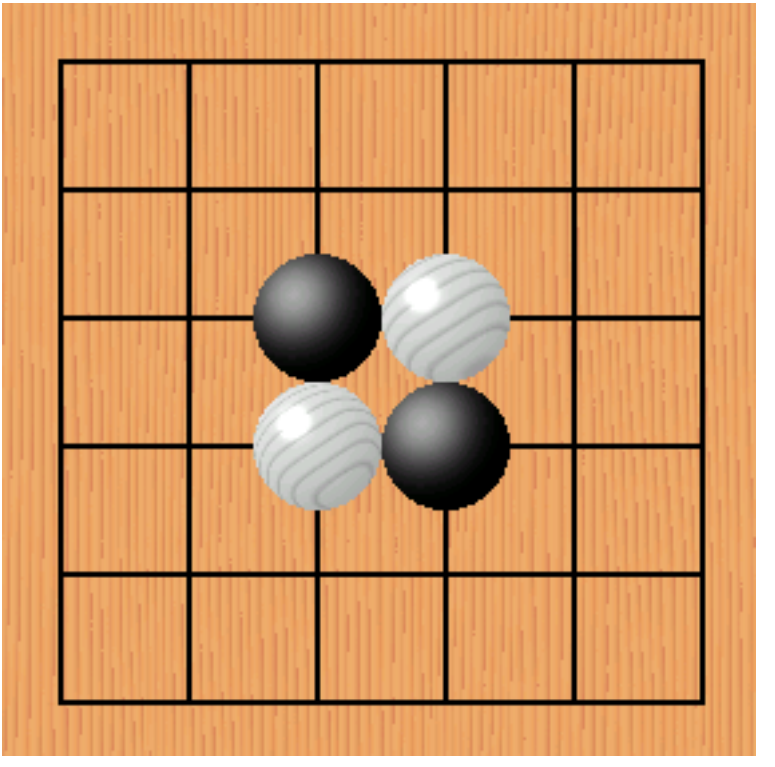


**Fig. 1.** The initial board for 6x6 AtariGo.

The rules are similar to Go: Black begins, Black and White alternate playing stones on the intersections of the board, strings of stones are stones of the same color that are linked by a line on the board. The number of empty intersections adjacent to the string is the number of liberties of the string. A string is captured if it has no liberty. For example in the Figure 1, all the strings have two liberties. A string that has only one liberty left can be captured by the other color in one move, it is in Atari, this is where the name of the game comes from. The goal of the game is to be the first player to capture a string of the opponent.

## 3 Admissible Number of Moves

In this section we start with explaining how it is possible to design admissible heuristics on the number of moves to win by relaxing the rules of the game. Then we give some example of admissible moves, and of the related admissible heuristics. The last subsection outlines possible optimizations of the computation of the designed heuristics.

### 3.1 Relaxation of the Rules of the Game

The admissible rules of a game are modified rules of the game. Usually, the admissible rules are more simple than the original ones. The rules of a game are admissible if the number of moves to win under these rules is always lower than the number of moves to win under the real rules, in any legal position.

An example of relaxation in the game of Go is to remove the forbidden moves. For example, relaxed rules where it is always legal to play on an empty intersection can be designed. Relaxations of the rules of a game can be automatically discovered by removing some conditions of the rules of the game represented in a logic language. However, with this method there are a large number of possible relaxations. It is not always easy to automatically find the useful relaxations out of all the possible ones. It can be easier in some games than in other. For example in Go, the number of liberties is easily deduced as an admissible heuristic on the number of moves to capture a string. For the 15-puzzle, the Manhattan distance can also be found with a relaxation of the rules of the game, just by removing the condition that the target tile has to be empty.

### 3.2 Admissible Moves

An admissible move is a move in a game with relaxed rules. The number of admissible moves required to win is always lower than the number of moves required to win in the real game.

For example in the game of Go, putting a stone on the liberty of a string is an admissible move. In the real game of Go, it is not always possible to play on the liberty of a string. For example it is not possible to remove a liberty if it is an eye, and if it is not the last liberty of the string. But if we relax the rules of Go, stating that it is always possible to put a stone on an empty intersection, we have admissible moves.

### 3.3 Admissible Number of Moves

The admissible number of moves is a lower bound on the number of moves needed to win the game. It is also the number of admissible moves needed to win the game. For example in the game of Go, the number of liberties of a string is an admissible heuristic of the number of moves in a row needed to capture the string.

In Philosopher's Football (Phutball) [1] threat search algorithms work very well. It is also possible to define simple admissible heuristics on the number of moves. A move in Phutball consists either in putting a white stone on an empty intersection, or jumping the ball (a black stone) over white stones. The game is over when the ball is on or behind the opponent goal line. A simple admissible heuristic on the number of moves is the length of the shortest line of empty intersections touching the goal line. The minimal number of moves in a row needed to win is half this length plus one, as there has to be at least one white stone every two intersections to move the ball to the goal line, and that moving the ball is the last move.

For the game of connection in Go, an admissible heuristic on the number of moves required to connect two strings is the length of the shortest path between the two strings. The length of the path being the number of empty intersections on the path plus the number of liberties that are not already on the path of the opponent strings that are on the path.

A refinement of these simple heuristics is to perform a tree search to find the minimum number of moves the attacker has to play when the defender is allowed to play one move, to play two moves separated by one or more moves by the attacker, etc... It is equivalent to a search algorithm, but it is faster than an Alpha-Beta as the Alpha-Beta is the limit of this sequence of trees: In Alpha-Beta the defender is allowed to play as many moves as the attacker. As the complexity of the search is exponential with the depth, these trees are computed much faster than the usual Alpha-Beta and they can stop search earlier and memorize useful information for move ordering as in the iterative deepening algorithm. We will come back to this refinement in the section on search.

### 3.4 Optimisation of the Computation of the Admissible Number of Moves

In Hex or in the connection game of Go, the admissible number of moves is the length of the shortest path between the two strings to connect. An optimization to the computation of the shortest path is to start searching the shortest path from the two strings to connect and to iteratively expand the perimeter around each string. It is equivalent to a bidirectionnal search, and it is faster than to expand from the first string until the second string is touched. It is also much faster that a brute force algorithm that would try all the possible moves.

An important speed-up come from the incremental updating of the heuristic. For the capture game of Go, it is not so simple to maintain the liberties incrementally from move to move. The algorithm used to incrementally update the

liberties is an union find algorithm. In our experiments, all the liberties of all strings are maintained incrementally each time a move is made and each time it is taken back. The admissible heuristic which is the minimum number of liberties over all the strings for each player is also maintained incrementally. The incrementality gives a substantial speed-up.

In our experiments, it is faster to maintain the liberties incrementally for the capture game than to recompute them when needed. We did not test the efficacy of incrementality for the connection game. We are not aware of any theory that could tell us in which games incrementality speeds computations up. Such a theory would be very useful.

## 4 Optimization of Gradual Abstract Proof Search with Abstract Knowledge on the Admissible Number of Moves

In games that have a large number of possible moves, and when playing few moves in a row often means a win, search algorithms based on threats can greatly outperform brute force Alpha-Beta. Go-Moku was solved by V. Allis using a search algorithm based on threats [2]. More recently, Abstract Proof Search [3] and Lambda Search [4] were designed and have outperformed basic Alpha-Beta search in the capture game of Go. The order of a position is the number of moves in a row that have to be played to win the game. A threat move leads to a position of order one. Abstract Proof Search uses abstract information to select the possible moves that can win a game at a given order.

### 4.1 Gradual Abstract Proof Search

Gradual Abstract Proof Search [5] is a refinement of Abstract Proof Search that solves the game of 6x6 AtariGo starting with a crosscut in the center. In AtariGo the first player to capture a string of stones wins. Abstract Proof Search selects MIN node moves using small depth bounded search. Lambda Search selects MIN node moves using order bounded search. Gradual Abstract Proof Search selects MIN node moves using both depth and order bounded search.

The gradual games that select moves at MIN nodes start with the letters 'ip'. Following the ip letters, a number gives the maximum number of attacker moves that can be played before winning the game. Then a sequence of numbers give the maximum order of each of these moves. For example the ip1 game finds moves that prevent the attacker from winning in one move. The ip2 game finds moves that prevent the attacker from winning in a depth two search tree. The ip2 game should be noted ip211, but as all the attacker moves in an ip2 game are always of order 1 it is simply noted ip2. The ip311 game finds moves that prevent a winning sequence of two consecutive direct threats followed by a winning move. All the attacker moves lead to positions of order one or less (winning position if the attacker moves and won positions). The ip4221 game finds moves that prevent the attacker from winning after two order 2 threats.

Figure 2 gives some examples of trees representing different gradual games. As in combinatorial game theory the two players are named Left and Right. The Left player is the attacker and the Right player is the defender. In these trees, a branch that goes on the left represents a Left move, and a branch that goes on the right represents some Right moves. Left branches are associated with winning moves for Left, and right branches are associated with the complete set of Right moves that can possibly prevent the win of the corresponding left branch (the left branch directly at the left of the right one, i.e. its sibling). All the leaves of the trees are positions won for Left. In order for the game to be verified, all Left moves have to be winning moves, and all Right moves have to be refuted by Left.

The tree labeled ip1 in the Figure 2 is the most simple one. The ip1 game is verified when the attacker can win in one move. A left branch that ends with a leaf node is always a winning move for the attacker. The ip2 tree represents a position where the attacker can win in at most two moves. The first left branch represents an attacker move that threatens to win. Below this first branch, there are two branches. The left one is the winning attacker move that executes the threat, and the right one represents all the defender moves that can potentially invalidate the threat. But all the defender moves are followed by a winning attacker move as symbolized by the last left branch. The other trees in the figure show some other games following the same graphical convention.

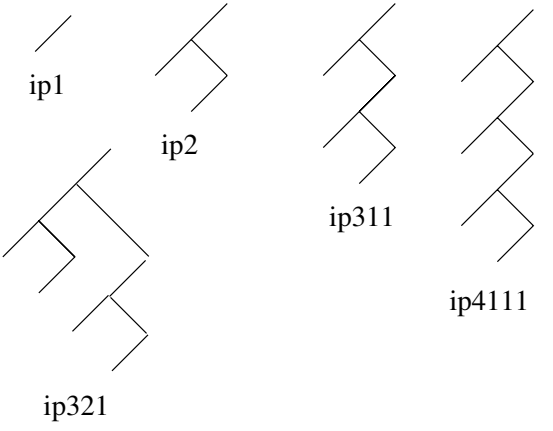



**Fig. 2.** Some gradual games.

### 4.2 Abstract Knowledge for Gradual Games

Verifying complex gradual games such as ip4221 can take a relatively long time. Abstract knowledge can be used to detect early that a gradual game cannot be

verified, for example when the number of admissible moves to win is greater than the order of the game. For example in AtariGo, it means that no order 2 game can be verified when the minimum number of liberties of all the defender strings is 3.

The abstract knowledge of order one is the knowledge to optimize the generation of possible moves when looking for a winning move. It consists in testing if there is an opponent string with only one liberty left. The attacker move generator returns the liberty if it is the case, and returns an empty set of moves when there is no such string. The defender move generator of order one first tries to capture an opponent string with only one liberty, and if there is none, it looks for friend strings with only one liberty. If there is one such string, it plays its liberty. If there is none, it returns an empty set of moves.

This order one abstract knowledge is very useful. An Abstract Gradual Proof Search with no abstract knowledge is impractical. We have stopped it after more than one hour of search. Whereas a simple order one abstract knowledge optimization makes it practical as can be seen in Table 1: 6x6 AtariGo with a cross cut in the centre is solved in less than 10 minutes.

The order two abstract knowledge for the attacker move generator consists in returning the liberty of a defender string if it has only one liberty, otherwise in returning an empty set if the minimum number of liberties of opponent strings is greater than two, else to return the liberty of a friend string with only one liberty, else to return the liberties of the opponent strings with two liberties, and if this last condition is not verified to return an empty set.

The order two abstract knowledge for the defender move generator is almost the symmetric of the attacker move generator, except when there is a defender string to save with two liberties. In this case the possible moves are the liberties of the defender string, the empty neighbors of the liberties of the defender string, the liberties and the empty neighbors of the liberties of the strings that have two liberties and which are also adjacent to the defender string, the liberties of the strings that have three liberties and which are also adjacent to the defender string and the liberties of the attacker strings that have two liberties.

The order three abstract knowledge for the attacker move generator is programmed in a similar way as the order two abstract knowledge, except that all the empty intersections that can be connected in two moves to the defender string are sent back when the defender string has only two liberties. Only the liberties are sent back when the defender string has three liberties.

There is no order three abstract knowledge for the defender move generator, it return all possible moves.

### 4.3 Experimental Results Quantifying the Usefulness of Abstraction

In order to estimate the usefulness of this abstract knowledge, we solved 6x6 AtariGo with Abstract Gradual Proof Search using different abstract knowledge orders. The results are given in the Table 1. The game is solved in 1 minute with the order three abstract knowledge, and in 10 minutes with the order one

**Table 1.** Search time with different abstractions orders.

| | Time (s) | | |
|---|---|---|---|
| Depth | *Order*1 | *Order*2 | *Order*3 |
| 1 | 0.06 | 0.02 | 0.00 |
| 2 | 17.86 | 4.23 | 2.41 |
| 3 | 18.06 | 4.04 | 2.98 |
| 4 | 54.86 | 11.56 | 5.56 |
| 5 | 52.95 | 9.59 | 5.43 |
| 6 | 114.71 | 21.66 | 10.19 |
| 7 | 127.42 | 23.25 | 12.85 |
| 8 | 174.98 | 38.28 | 19.09 |
| 9 | 21.07 | 8.15 | 3.89 |
| 10 | 3.16 | 0.68 | 0.25 |
| Total | 585.13 | 121.46 | 62.65 |

abstract knowledge. We did not report the results for the solution with no abstract knowledge as it took too much time. The maximum gradual game needed to solve 6x6 AtariGo is ip4221. At each MIN node, the ip1, ip2, ip311, ip4111, ip51111, ip4121, ip4211, ip4221 games are checked. If at least one game is verified, the intersection of all the sets of moves sent back by the verified games is performed and the moves in the intersection set are tried for the defender.

**Table 2.** Average time used to verify gradual games for different abstract orders.

| | Average time ($\mu$s) | | |
|---|---|---|---|
| Game | *Order*1 | *Order*2 | *Order*3 |
| ip1 | 2 | 5 | 5 |
| ip2 | 328 | 66 | 136 |
| ip311 | 1165 | 137 | 137 |
| ip4111 | 1751 | 253 | 220 |
| ip51111 | 1763 | 187 | 148 |
| ip321 | 36877 | 8476 | 4614 |
| ip4121 | 63737 | 7843 | 3809 |
| ip4211 | 40451 | 9477 | 4878 |
| ip4221 | 196836 | 40368 | 20661 |

We have also output the average time used for the verification of the different gradual games. The results are given in Table 2. As can be expected the abstract knowledge of order three is quite useful for the more complex gradual games. For the ip4221 game, using abstractions of order 3 is 10 times faster than using abstractions of order 1, and twice as fast as abstractions of order 2.

The overall Alpha-Beta that calls the gradual games at MIN nodes, uses transposition tables, iterative deepening, two killer moves, the history heuristic and the difference between the minimum number of liberties of the attacker and the minimum number of liberties of the defender as an evaluation function.

The verification of the games definitions in Abstract Proof Search and Gradual Abstract Proof Search can be related to the progressive admissibility of the depth of the win. We have used so far as an admissible heuristic the number of moves in a row of the attacker color needed to win the game. If instead, we refine the heuristic by taking into account only one defender move. We still have an admissible heuristic on the depth of the win, but it is usually higher as the defender is allowed to play one move before all the attacker moves are played. If the admissible heuristic with one move for the defender gives a number greater than the maximal depth allowed, then we can cut. The number of allowed defender moves can be progressively increased until it is equal to the number of attacker moves or the depth of the search is too high. This progressive increase of the admissible heuristic costs one defender move at each step. The cost of the search is exponential in the number of moves. The final tree is the Alpha-Beta tree. This progressive refinement of the admissible heuristic toward the real Alpha-Beta is very close to what is performed when verifying the gradual games definitions as the reader can verify.

Currently, we do not use heuristics for move ordering when verifying the gradual games. It would make the search faster to use transposition tables and the killer move heuristic in the gradual games.

## 5 Automatic Generation of Rules

Retrograde analysis of patterns has been succesfully applied to the 15-puzzle [6] and to the Rubik's cube [7] to improve the accuracy of admissible heuristics. In our application to the game of Go, we rather use admissible heuristics to generate safe rules associated with external conditions by retrograde analysis [8]. A rule is a rectangular pattern associated with conditions related to the number of liberties of the objects in the pattern. The admissible number of moves is used to reduce the amount of learned rules, to generate more general rules and to ensure the validity of the generated rules. Hundreds of thousands of rules have been generated for making one eye, making two eyes and connecting strings. They give a large speed-up for our problem solver as they enable the solver to detect eyes and life many moves in advance.

In an automatically generated rule, the conditions associated with the attacker are always admissible conditions. For example, if the attacker has to be able to resist two defender moves for the capture of a string, the associated condition is that the minimum number of liberties of the string at the exterior of the pattern is three. On the contrary, the conditions associated with the defender are always upper bounds of the real value. For example, if a defender string has to be captured in one move when it has no more liberties at the interior of the pattern, the associated condition is that the string has one liberty or no

liberty at the exterior of the pattern. There are never conditions on more than one liberty for the defender, as a string with two external liberties can have two eyes, and can possibly never be captured whatever the number of moves of the attacker.

The conditions for the defender strings are the maximal number of moves to remove all his liberties, and the conditions for the attacker strings are minimal number of moves to remove them. This ensures that the attacker will always be able to verify the conditions, whatever the real values are. On the contrary of admissible heuristics in one player games where there are only lower bounds on the number of moves, here we see that we need both lower and upper bounds on the number of moves.

It would be interesting to relate the time used to solve problems with the size of the generated databases and see if the same behavior is observed as in single agent search [9]. The condition that the number of external liberties of the defender is always less than one can sometimes prevent from learning useful rules. A refinement of the heuristic to take into account situations where the maximum number of moves to capture a defender string is greater than one would enable our system to generate more complex rules.

## 6 Automatic Generation of Heuristics

Introspect is a partial evaluator specialized on games [10]. It can automatically generate programs that verify the gradual game definitions described in the section on search. It uses the rules of the game written in first order logic to unfold the gradual games definitions and obtains efficient programs for gradual games. It is able to discover by partial evaluation the admissible heuristics on the number of moves. For example, in the generated programs for the capture game of Go, the number of liberties of the string is tested at first to see if the string can be captured a given number of moves ahead. The generated programs are also very selective on the moves to consider. They have knowledge very similar to the knowledge on relevant moves described in the section on search.

An important issue when using partial evaluation to unfold the rules of a game is the set of predicates used to represent the rules. Different rules representations can generate very different programs. For example, if some of the rules used to specialize the capture game contain the two commonly occuring conditions 'liberties(X, N), N > 2' it is worse than containing the condition 'minliberties(X, 2)'. In the first case, the specializer will unfold the definition of liberties, and there are many rules to update liberties after a move. In the second case, the number of rules to compute the minimal number of liberties is much less. As the specializer will unfold the rules of the game as many times as there are moves in a gradual game, it will generate a very large number of rules in the first case, and only a few rules in the second case.

A related problem is the unfolding of recursive predicates. The number of liberties is a recursive function, and it is well known that unfolding recursive function has to be done very cautiously, and sometime is not to be done at all.

In the case of the number of liberties, the unfolding leads to an explosion of the number of generated rules and to worse generated programs. So the number of liberties should not be unfolded.

Given these warnings on the use of partial evaluation to generate game knowledge, it is possible to use partial evaluation to generate clever programs that select relevant moves and find admissible heuristics on moves. The unfolding on the rules of the game of the dumb algorithm that plays all the possible moves n times in a row to see a win can generate a very selective and efficient algorithm for generating moves. Another possible use of partial evaluation is to find fast way to compute the heuristics when they are given. For example, it could find a fast way to compute the shortest path between two strings given a simple and inefficient algorithm.

A more ambitious goal for Introspect is to discover more accurate heuristics. Given a simple heuristic such as the number of liberties of a string in the capture game, it could transform the definition of the heuristic to find another one which always gives a greater value. For example, a more accurate admissible heuristic for the capture game is to add the number of independent protected intersections minus one. An intersection is protected if the opponent is captured when he plays on it. It is possible to find this heuristic and some other by unfolding the definition of the heuristic with the MinMax algorithm and the rules of the games. Once this unfolding is performed, many rules are generated that give all the cases when the heuristic has underestimated the number of moves. Out of all this rules, the most simple and general can be kept. A similar method to generate admissible heuristics is to remove conditions inside the rules of the game [11,12]. The two methods overlap, but a combination of the two might give better results than each of them.

## 7 Conclusion and Future Work

We have given experimental evidence that admissible heuristics on moves in two-player games account for a large speed-up for threat search algorithms. These algorithms are already much faster than basic Alpha-Beta for the games of AtariGo, Phutball and others games with frequent low order threats. The admissible heuristics we have used can also improve a depth-bounded Alpha-Beta to stop search earlier. We feel that some progress can still be made in the accuracy of these heuristics so as to improve the efficacy of current threat based search algorithms.

Admissible heuristics on the number of moves are also required in the generation of rules by retrograde analysis. Using them, our system reduces the amount of learned rules, generates more general rules and ensures the validity of the generated rules.

Some work is still needed to understand the reasons why incrementality of the computation of the admissible heuristics works well for some heuristics in some games but not in others. Improving the accuracy of our current admissible heuristics on the number of moves could speed-up our search algorithm by orders

of magnitude. We are interested in experimenting threat search algorithms with different games and different admissible heuristics. Introspect, our partial evaluation system can be used to automatically generate accurate heuristics from simple ones. A special attention has to be given to the representation of the rules of the game, as different representations can lead to very different qualities of generated programs. Introspect can also be used to generate programs that can compute a given heuristic faster. In some games where it is not so easy to write admissible heuristics, it has been used as a program writing assistant that writes long, complex and efficient programs by unfolding the definition of the gradual games on the rules of the game.

## References


1. Conway, J.H., Berlekamp, E., Guy, R.K.: Philosopher's football. In: Winning Ways. Academic Press (1982) 688–691
2. Allis, L.V., van den Herik, H.J., Huntjens, M.P.H.: Go-moku solved by new search techniques. Computational Intelligence **12** (1996) 7–23
3. Cazenave, T.: Abstract proof search. In Marsland, T.A., Frank, I., eds.: Computers and Games. Volume 2063 of Lecture Notes in Computer Science., Springer (2002) 39–54
4. Thomsen, T.: Lambda-search in game trees - with application to go. In Marsland, T.A., Frank, I., eds.: Computers and Games. Volume 2063 of Lecture Notes in Computer Science., Springer (2002) 19–38
5. Cazenave, T.: Gradual abstract proof search. In: Proceedings of RFIA, Angers, France (2002)
6. Culberson, J., Schaeffer, J.: Pattern databases. Computational Intelligence **14(4)** (1998) 318–334
7. Korf, R.E.: Recent progress in the design and analysis of admissible heuristic functions. In: AAAI/IAAI. (2000) 1165–1170
8. Cazenave, T.: Generation of patterns with external conditions for the game of go. In van den Herik, H., Monien, B., eds.: Advance in Computer Games 9. Universiteit Maastricht, Maastricht (2001) 275–293 ISBN 90 6216 566 4.
9. Holte, R., Hernadvolgyi, I.: Experiments with automatically created memory-based heuristics. In: SARA-2000, Lecture Notes in Artificial Intelligence N° 1864 (2000) 281–290
10. Cazenave, T.: Synthesis of an efficient tactical theorem prover for the game of go. ACM Computing Surveys **30** (1998)
11. Knoblock, C.A.: Automatically generating abstractions for planning. Artificial Intelligence **68** (1994) 243–302
12. Prieditis, A., Davis, R.: Quantitatively relating abstractness to the accuracy of admissible heuristics. Artificial Intelligence **74** (1995) 165–175

# Dynamic Bundling: Less Effort for More Solutions

Berthe Y. Choueiry and Amy M. Davis

Constraint Systems Laboratory
Department of Computer Science and Engineering
University of Nebraska-Lincoln
{choueiry | amydavis }@cse.unl.edu

**Abstract.** Bundling of the values of variables in a Constraint Satisfaction Problem (CSP) as the search proceeds is an abstraction mechanism that yields a compact representation of the solution space. We have previously established that, in spite of the effort of recomputing the bundles, dynamic bundling is never less effective than static bundling and non-bundling search strategies. Objections were raised that bundling mechanisms (whether static or dynamic) are too costly and not worthwhile when one is not seeking *all* solutions to the CSP. In this paper, we dispel these doubts and empirically show that (1) dynamic bundling remains superior in this context, (2) it does not require a full lookahead strategy, and (3) it dramatically reduces the cost of solving problems at the phase transition while yielding a bundle of multiple, robust solutions.

## 1 Introduction

Many problems in engineering, computer science, and management are naturally modeled as Constraint Satisfaction Problems (CSPs), which is, in general, **NP**-complete. Backtrack search remains the ultimate mechanism for solving such problems. One important mechanism for enhancing the performance of search is the exploitation of symmetries in the problem or a particular instance of it. From a practical perspective, the exploitation of symmetry can be used both to reduce the size of the search space and, more importantly, to represent the *solution space* in a compact manner by identifying families of qualitatively equivalent solutions, as we argued which is useful in practical applications [6].

In this paper we study the following two issues: (1) the combination of the dynamic computation of symmetries during search with the currently most popular lookahead strategy, and (2) the effect of symmetry detection on the presence and severity of the phase-transition phenomenon believed to be inherent to **NP**-complete problems. Our results are two fold. First, in accordance with [11], we dispel the growing myth that the aggressive lookahead strategy known as Maintaining Arc Consistency (MAC) [17] is always beneficial. Second, we establish that the dynamic detection and exploitation of symmetries during search, which results in multiple robust solutions, does not impede the performance of search

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 64–82, 2002.

but is actually a cost-effective tool for dramatically reducing the cost peak at the phase transition, possibly the most critical phenomenon challenging the efficient processing of combinatorial problems in practice.

## 2 Background and Motivation

Glaisher [13], Brown et al. [3], Fillmore and Williamson [9], Puget [16] and Ellman [8] proposed to exploit *declared* symmetries among values in the problem to improve the performance of search. The first four papers considered *exact* symmetries only, and the latter proposed to include also necessary and sufficient *approximations* of symmetry relations. Freuder [10] introduced a classification of various types of symmetry, which he called interchangeability. While all prior approaches focused on declared symmetry relations, Freuder proposed an efficient algorithm that *discovers* an exact but local form of interchangeability, neighborhood interchangeability (NI). NI partitions the domain of a given variable into a set of equivalence classes of values. Haselböck [14] simplified NI into a weaker form that we call *neighborhood interchangeability according to one constraint* ($\mathrm{NI}_C$). Further, he showed how to exploit $\mathrm{NI}_C$ advantageously during backtrack search by generating solution *bundles*. Every solution bundle is a set of robust solutions [12]: Any value for a variable can be safely replaced by another value for the variable taken from the same domain bundle without altering the assignments of the remaining variables. Since the strategy devised by Haselböck computes domain partitions in a pre-processing step prior to search, we call this strategy *static bundling* and denote it NIC-FC. We proposed [7] a weak form of NI, namely *neighborhood partial interchangeability* (NPI) that can be controlled to compute interchangeability anywhere between, and including, NI and $\mathrm{NI}_C$, see Fig. 1. In [1,2] we proposed to recompute NPI relations *dynamically* during


**Fig. 1.** *Three types of neighborhood interchangeability.*

search yielding a new type of interchangeability we called *dynamic neighborhood partial interchangeability* (DNPI). We call this *dynamic bundling* and the search strategy DNPI-FC. While restricting our investigations to forward checking (FC) as a lookahead strategy during search, we established the superiority of dynamic bundling (DNPI-FC) over both static bundling (NIC-FC) and non-bundling search (FC) in terms of the criteria that assess the search effort and the 'compaction' of the solution space. These *theoretical* results hold when looking for all solutions and using a static variable ordering.

## 3 Our Study and Expectations

There has been a misconception that the cost of bundling in general and that of dynamic bundling in particular require too much overhead when one is only looking for a *first* solution. Indeed, the following two erroneous impressions prevailed: (1) when looking for a first solution, bundling–whether static or dynamic–is not worth the effort, and (2) all the more reason, dynamic bundling is an overkill. We showed empirically[1] that the above stated superiority of dynamic bundling holds almost always in practice when looking for one solution and under various ordering heuristics [2]. For lack of space, we cite here only a few of the ones we investigated: (1) *Static ordering* (`SLD`): Variables are ordered statically before search according to the least domain heuristic. (2) *Dynamic variable ordering* (`DLD`): Variables are ordered dynamically during search according to the same heuristic. And (3) *Dynamic variable-value ordering:* In the context of bundling, a value is in fact a bundle. For this purpose, we proposed a new strategy, Least Domain-Max Bundle [2] (`LD-MB`) that chooses, dynamically during search, the variable of smallest domain (as in `DLD`) and, for this variable, the largest bundle in its domain. This heuristic[2] proved to be superior to other heuristics [5].

The surprising performance of dynamic bundling is explained by the fact that, while bundling partial solutions, it also *factors out the no-goods*, thus effectively pruning the search space. The two questions below remained unanswered:

1. How well does dynamic bundling combine with the most aggressive and popular, lookahead strategy MAC?
2. How does dynamic bundling affect the spike of the problem-solving cost at the phase transition identified in [4] and extensively studied in [15]? The cost and behavior of bundling on the cross-over point (i.e., critical area of the order parameter [4]) is the ultimate for its practical utility.

We concentrate on finding the *first* solution (bundle) *where we expect the cost of bundling to hinder seriously the performance of search.* To answer these questions, we conduct the experiments summarized below:

| **Question** | **Values reported** | **Ordering** |
|---|---|---|
| Dynamic bundling: MAC vs. FC | Ratio of DNPI-MAC to DNPI-FC | `SLD`, `DLD`, `LD-MB`, Fig. 3 |
| Effects of bundling on the phase transition | DNPI-FC, DNPI-MAC<br>NIC-FC, (non-bundling) FC | `SLD`, Fig. 4<br>`DLD`, Fig. 5<br>`LD-MB`, Fig. 6 |

Below, we discuss the context of these two questions and list our expectations. Then, in Section 4, we present our experiments, provide a list of observations that summarize our findings, and finally discuss these results in detail.

[1] Under a wide variety of testing conditions and for problems in which we finely controlled the amount of interchangeability embedded in a given problem instance.

[2] Note that for finding all solutions `LD-MB` collapses to `DLD`.

### 3.1 Lookahead Strategies

Sabin and Freuder [17] introduced a procedure to Maintain Arc Consistency (MAC), and advised to use this full-lookahead strategy instead of the popular partial-lookahead strategy known as forward checking (FC). While FC propagates the effects of a variable assignment to connected future variables, MAC propagates these effects over the entire remaining (future) constraint network. This stronger filtering of MAC has the potential to be particularly advantageous when coupled with dynamic bundling. We call this new search strategy DNPI-MAC. A similar study was independently conducted by Silaghi et al. in [18], who coupled MAC with the Cross Product Representation (CPR)[3]. They consider two different implementations of MAC, and show that they are comparable. They test *only* CPR and *only* MAC omitting (1) whether or not MAC is beneficial and (2) a comparison of CPR with static and non-bundling search strategies. We seek to quantify the value of adding MAC to dynamic bundling.

We first study dynamic bundling in combination with MAC and with FC under various ordering heuristics. In Section 3.2 we further compare them (i.e., DNPI-MAC and DNPI-FC) to non-bundling and static bundling strategies with the goal of studying their behavior at the phase transition. We anticipate that the integration of DNPI and MAC will fulfill the expectations discussed below.

**Expectation 1.** *Since MAC performs a stronger pruning than FC, DNPI-MAC should not visit more nodes than DNPI-FC does.*

Indeed any value that is pruned by MAC and not pruned by FC is an additional node that FC examines. Further, it is guaranteed to fail and it results in extra useless work for the FC search strategy. We suspect that Expectation 1 could stand as a theorem *under static ordering*. It is supported by strong empirical evidence in Section 4.2, Fig. 3 and Observation 5, which relate average values over a pool of 6040 random problems. However, a careful examination of the results uncovered a *single* exception that we have not yet resolved.

**Expectation 2.** *DNPI-MAC should generate larger bundles than DNPI-FC and every bundle of DNPI-FC should be a subset of a bundle of DNPI-MAC under the same static variable ordering. The following expression should hold:*

$$\texttt{SB}(\textit{FC}) \geq \texttt{SB}(\textit{NIC-FC}) \geq \texttt{SB}(\textit{DNPI-FC}) \geq \texttt{SB}(\textit{DNPI-MAC}) \qquad (1)$$

*where* SB *is the number of solution bundles found. When finding only the first solution, Equation (1) suggests a statement about the First Bundle Size (*FBS*) (when* SB *is small, bundles are large). Thus we anticipate the following:*

$$\texttt{FBS}(\textit{FC}) \leq \texttt{FBS}(\textit{NIC-FC}) \leq \texttt{FBS}(\textit{DNPI-FC}) \leq \texttt{FBS}(\textit{DNPI-MAC}) \qquad (2)$$

**Expectation 3.** *Because of the above, we are tempted to infer that DNPI-MAC should be computationally cheaper than DNPI-FC and perform better bundling.*

[3] Silaghi et al. erroneously claim that CPR was proposed as DNPI. We showed independently that CPR and DNPI yield the same bundling while DNPI visits fewer nodes. The difference is polynomially bounded, as suggested by a reviewer.

### 3.2 Phase Transition

Cheeseman et al. [4] presented empirical evidence of the existence of a phase transition phenomenon in the cost of solving **NP**-complete problems when varying an order parameter. In particular, they showed that the location of the phase transition and its steepness increase with the size of the problem[4], thus yielding a new characterization of this important class of problems. We now examine the effect of bundling (statically and dynamically) on the phase transition. Because problems at the cross-over point are acknowledged to be probabilistically the most difficult to solve, determining the performance of our algorithms in this region is important. So far, we have not found a scenario where dynamic bundling is any hindrance to the performance of search, both for all solutions and for one solution. It is further aided by dynamic variable ordering, but *not* by MAC as discussed in Observations 5 and 3. In these experiments with the phase transition, we prepare the most adverse situation that we know of for our algorithms. However, given the ability of dynamic bundling in pruning 'bundled' no-goods, we anticipate the following:

**Expectation 4.** *Dynamic bundling will significantly reduce the steepness of the phase transition, that is, the problems in the phase transition will be easier to solve with dynamic bundling.*

## 4 Experiments

In this section, we examine the above expectations through empirical tests. We used the random generator for binary CSPs described in [2]. This generator allows us to control the level of interchangeability embedded in an instance of a CSP by controlling the number of equivalence classes of values induced by every constraint on the domain of one of the variables in its scope. We call this number the induced domain fragmentation `IDF`. Fig. 2 shows a constraint $C$ with an `IDF`=3. For each measurement point, we generated 20 instances with

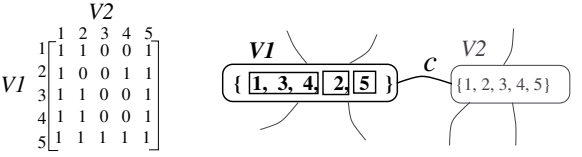



**Fig. 2.** *Left:* Constraint as a binary matrix. *Right:* Domain of $V_1$ partitioned by $C$.

the following characteristics: number of variables $n = 20$; domain size $a = 10$; induced domain fragmentation `IDF` $= [2, a]$ by a step of 1; tightness (ratio of

[4] Problems in **P** do not in general contain a phase transition, or if they do, the cost at the transition bounded and is not affected by an increase of problem size [4].

number of allowed tuples over that of all possible tuples) of any constraint $t = [0.15, 0.85]$ with a step of 0.05; and constraint probability (ratio of the number of constraints in the CSP over that of all possible constraints) $p = 0.5$ and 1.0.

In order to reduce the duration of our experiments to a reasonable value, we chose to make *all* problems arc-consistent (AC-3) before search begins. This is done uniformly in all experiments and for all strategies and does not affect the quality of our conclusions. We compute, at each data point, both the average and the median values of the following *evaluation criteria*: number of nodes visited (`NV`); number of constraint checks (`CC`); CPU time in msec with a clock resolution of 10 msec; and size of the first bundle found (`FBS`). The horizontal axis denotes various tightness values $t = 0.15 \cdot 0.4$. Problems beyond $t = 0.4$ were determined unsolvable by the preprocessing step and thus required no search. We notice that the average and median curves almost always have the same shapes (except for one case discussed below). Our experiments yield a number of observations summarized in Section 4.1 and discussed in detail in Sections 4.2 and 4.3.

### 4.1 List of Observations

We first state a general observation (Observation 1), then observations regarding the effect of lookahead strategies (Observations 5 to 5) and finally observations about the effect on the phase transition (Observations 6 to 16).

**Observation 1.** *The curves for constraint checks* (`CC`) *and CPU time are often similar in shape and differ from that for* `NV`*, suggesting that constraint checks dominate the computational cost in our implementation.*

**Observation 2.** *DNPI-MAC always visits fewer nodes* (`NV`) *than DNPI-FC, in confirmation of Expectation 1.*[5]

**Observation 3.** *DNPI-MAC in general requires more constraint checks* (`CC`) *than DNPI-FC. This effect always holds under dynamic orderings* (`DLD` *and* `LD-MB`) *where DNPI-MAC performs particularly poorly.*

When constraint checks, and not nodes visited, dominate computation cost, DNPI-FC performs better than DNPI-MAC. Thus, contrary to Expectation 3, the advantage in fewer nodes visited does not translate into saved time, yielding:

**Observation 4.** *Either because of its high cost in CPU time (which, in our implementation, seems to reflect more the effort spent on checking constraints than that spent on visiting nodes), or because the advantages of DNPI-MAC in terms of* `NV` *does not balance out the loss for constraint checks, DNPI-MAC is more costly than DNPI-FC. This tendency is aggravated under dynamic orderings where the performance of MAC further deteriorates.*

[5] This observation holds for the *average* values reported in our graphs of Fig. 3, however we detected a single anomaly as mentioned in Section 3.1.

**Observation 5.** *The solution bundle found by DNPI-MAC is in general not significantly larger than that found by FC and does not justify the additional computational cost.*

**Observation 6.** *The magnitude and steepness of the phase transition increases proportionally with p, in accordance with the experiments reported in* [15].

**Observation 7.** *Although dynamic bundling does not completely eliminate the phase transition, it dramatically reduces it.*

**Observation 8.** `DLD` *orderings are generally less expensive than* `SLD` *orderings for all search strategies and yield larger bundles.*

**Observation 9.** `DLD` *orderings are also generally less expensive than* `LD-MB` *for dynamic bundling but similar for static bundling. However,* `LD-MB` *orderings produce larger bundles.*

**Observation 10.** `LD-MB` *orderings are generally less expensive than* `SLD` *orderings for all search strategies and yield larger bundles.*

**Observation 11.** *The bundle sizes of all bundling strategies are comparable, thus their respective advantages are better compared using other criteria.*

**Observation 12.** *DNPI-MAC is effective in reducing the nodes visited* `(NV)` *at the phase transition.*

**Observation 13.** *DNPI-MAC does not significantly reduce the overall cost at the phase transition.*

**Observation 14.** *In static orderings, the reduction of the phase transition due to the use of MAC seems to be more significant than that due to the use of dynamic bundling.*[6]

**Observation 15.** *Static bundling ($NI_C$) is expensive in general and we identify no argument to justify using it in practice. Further, under dynamic orderings, its high cost extends beyond the critical area of the phase transition to the point of almost concealing the spike.*[7]

**Observation 16.** *In dynamic orderings, DNPI-FC is a clear 'champion' among all strategies with regard to cost (i.e., constraint checks and CPU time).*

[6] We stress that this effect is reversed in dynamic orderings.

[7] The high cost of $NI_C$ in the zone of 'easy' of problems is linked to the overhead of pre-computing interchangeability prior to search while many solutions exist.

### 4.2 Data Analysis: MAC vs. FC

As stated above, we report the results for finding one solution bundle. Fig. 3 shows the *ratio* of the values of the evaluation criteria for MAC versus FC under dynamic bundling and for the three ordering heuristics SLD (⧫), DLD (□), LD-MB (×). For values above 1, the value of DNPI-MAC is higher than the value of FC, and vice versa. Below we discuss each row:

*Nodes visited (row 1):* The value of the ratio is consistently below 1 across ordering strategies, IDF values, and $p$ values. This indicates that MAC always visits fewer nodes than FC and supports Observation 5. Note that this effect becomes more pronounced as the constraint probability increases from 0.5 (left column) to 1.0 (right column) and as tightness increases (shown by the downward slope of the lines).

*Constraint checks (row 2):* The non-null values (except for a few black diamonds that we discuss below) are all above 1, indicating that FC is superior to MAC (Observation 3). Now let's look at the few cases where DNPI-MAC outperforms DNPI-FC (ratio $< 1$). This sometimes happens, almost exclusively under the static ordering SLD (⧫). It happens for: (1) $p$=0.5, $t$ is large, and for all values of IDF, and (2) $p$=1.0 and IDF is small. This provides an interesting result, because DNPI-MAC is sensitive to IDF where DNPI-FC is insensitive (specifically, when constraint probability is high). As IDF increases, we see that DNPI-MAC loses the edge it had when more interchangeability was present. Note also that when using dynamic variable ordering such as DLD (□) or LD-MB (×), DNPI-FC is a clear winner over DNPI-MAC. When we use dynamic variable ordering, DNPI-MAC checks from 3 to 13 times as many constraints as DNPI-FC. This supports Observation 3 and is a *clear indication of an expense in MAC that is not compensated by dynamic bundling.*

*CPU time (row 3):* The advantage of DNPI-FC over DNPI-MAC noted in the constraint checks is reinforced by the CPU time data. Thus the use of DNPI-MAC (except for a few cases, mostly in SLD) is *detrimental* to the performance of search despite the savings of nodes visited (Observation 4).

*Size of first bundle (row 4):* Finally, we look at the bottom row of Fig. 3 to check whether or not the additional propagation effort of MAC benefits the size of the bundle found (Observation 5). We see that, and in accordance with Expectation 2, DNPI-MAC does generate slightly larger bundles than DNPI-FC. For $p$=0.5, the bundle comparisons huddle mostly just above 1. This means that the bundle sizes are comparable, with DNPI-MAC generally producing bundles that are just a little bit larger. We note two extreme behaviors:

1. At IDF=2, $t$=0.40, the bundle produced by DNPI-MAC is fifteen times larger than that of DNPI-FC (⧫). Additionally, this much larger bundle took less time to find. This demonstrates and justifies the advantage of DNPI-MAC. Note however that this extent of divergence between DNPI-MAC and DNPI-FC does not hold when IDF$>$ 2. Indeed, for $p$=1.0, the bundles of DNPI-MAC are never more than three times larger than those of DNPI-FC, but are frequently smaller (especially under dynamic variable ordering).

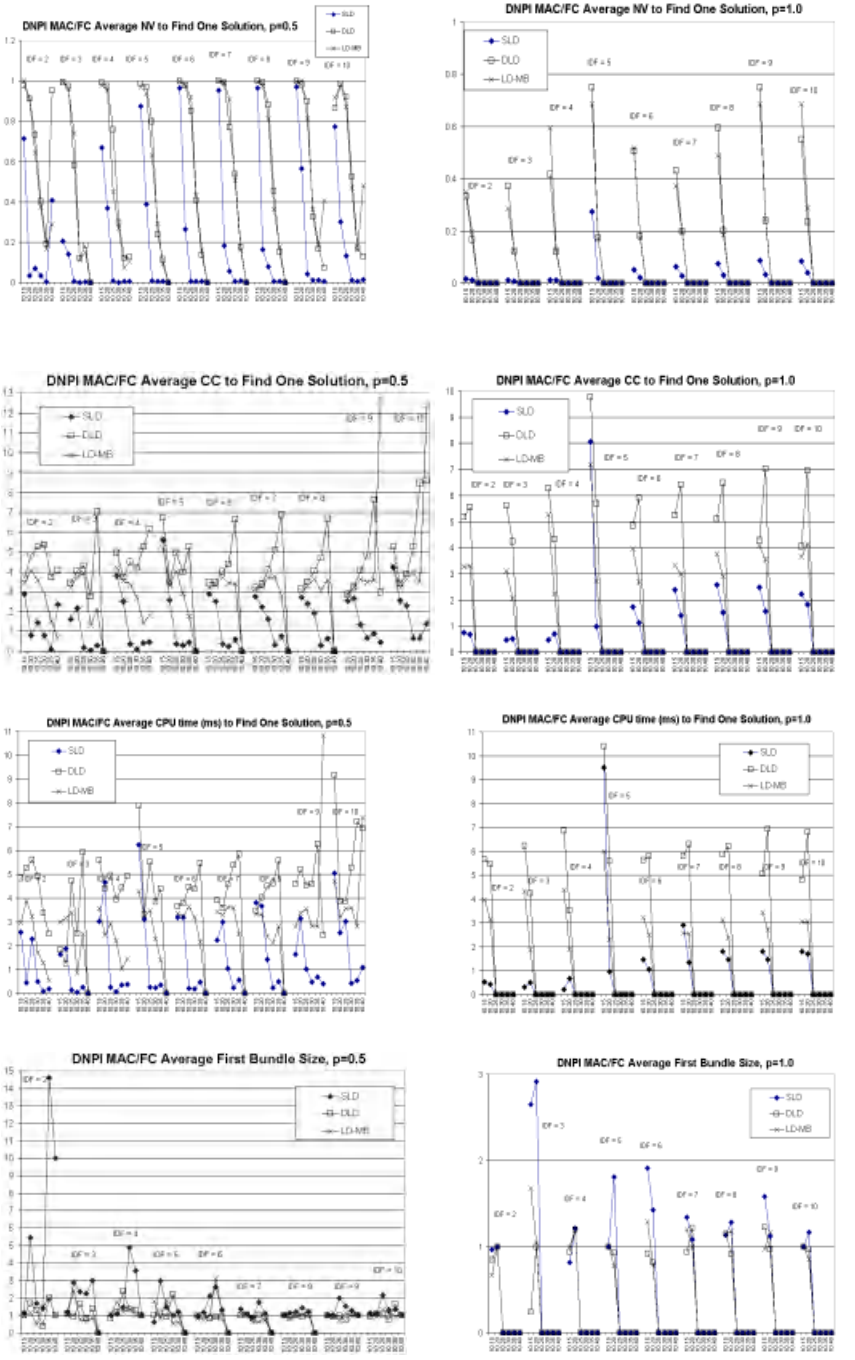



**Fig. 3.** *Comparing MAC and FC in the context of DNPI.*

2. At IDF=4 and $t$=0.15, the bundle produced by DNPI-MAC is smaller than that of DNPI-FC (⧫). This is the only major opposition to our Expectation 2. (There are other small exceptions, where DNPI-FC produces a bundle that is 1 or 2 solutions larger than DNPI-MAC's bundle. We traced all these exceptions to the non-deterministic value ordering.) We traced this violation of Expectation 2 to a single problem where DNPI-MAC found a bundle of size 84 and DNPI-FC found a bundle of size 168. This happens for the single problem mentioned above, which we cannot yet justify.

*Conclusions of the data analysis:* The cost of DNPI-MAC is neither systematically nor predictably worthwhile, except possibly under SLD ordering. This tendency becomes even stronger under dynamic orderings DLD and LD-MB.

### 4.3 Data Analysis: Dynamic Bundling at the Phase Transition

These charts are arranged in Fig. 4, 5, and 6 for the ordering heuristics SLD, DLD, and LD-MB, respectively. Each graph shows four search strategies, namely non-bundling FC (△), static-bundling NIC-FC (×), dynamic bundling with forward checking DNPI-FC (⧫), and dynamic bundling with Maintaining Arc Consistency DNPI-MAC (□). Each figure has two columns: left for $p$=0.5 and right for $p$=1.0. We first state some global observations over the experiments as a group and across ordering heuristics. Then we examine the data for each of the three ordering heuristics in this order: SLD, DLD, then LD-MB. In each graph, we pay particular attention to the relative behavior of the search strategies at the phase transition, demonstrated here by the presence of a 'spike' in nodes visited, constraint checks and CPU time.

*Global observations:* The comparison of the left and right charts in Fig. 4, 5, and 6, confirms the findings in [15] on how the slope and importance of the phase transition augment with the constraint probability (Observation 6). The examination of all 24 graphs at the phase transition peak confirms that dynamic bundling dramatically reduces its magnitude (Observation 7). Finally, the comparison of graphs for CPU time and bundling power, interpreted as first bundle size across ordering strategies, shows that DLD is consistently an excellent ordering unless one is specifically seeking larger bundles at the expense of a slight increase in cost in which case LD-MB is justified (Observations 8, 9, and 10).

#### 4.3.1 DNPI at Phase Transition: Static Ordering

Recall from Observation 4 and Section 4.2 that DNPI-MAC performs best under SLD ordering. Under dynamic orderings (DLD and LD-MB), it is non-competitive.

*Nodes visited (NV) with SLD (row 1):* Fig. 4 shows that strategies based on dynamic bundling (□ and ⧫) expand in general fewer nodes than strategies based on non-bundling (△) or static bundling (×) (Observation 7). A careful examination shows that the phase transition is indeed present for DNPI-FC (to some extent) and for NIC-FC and FC (to a large extent) but seemingly absent in DNPI-MAC (Observation 12). Recall that MAC almost always visits fewer nodes than DNPI-FC, which is guaranteed to visit fewer nodes than the others

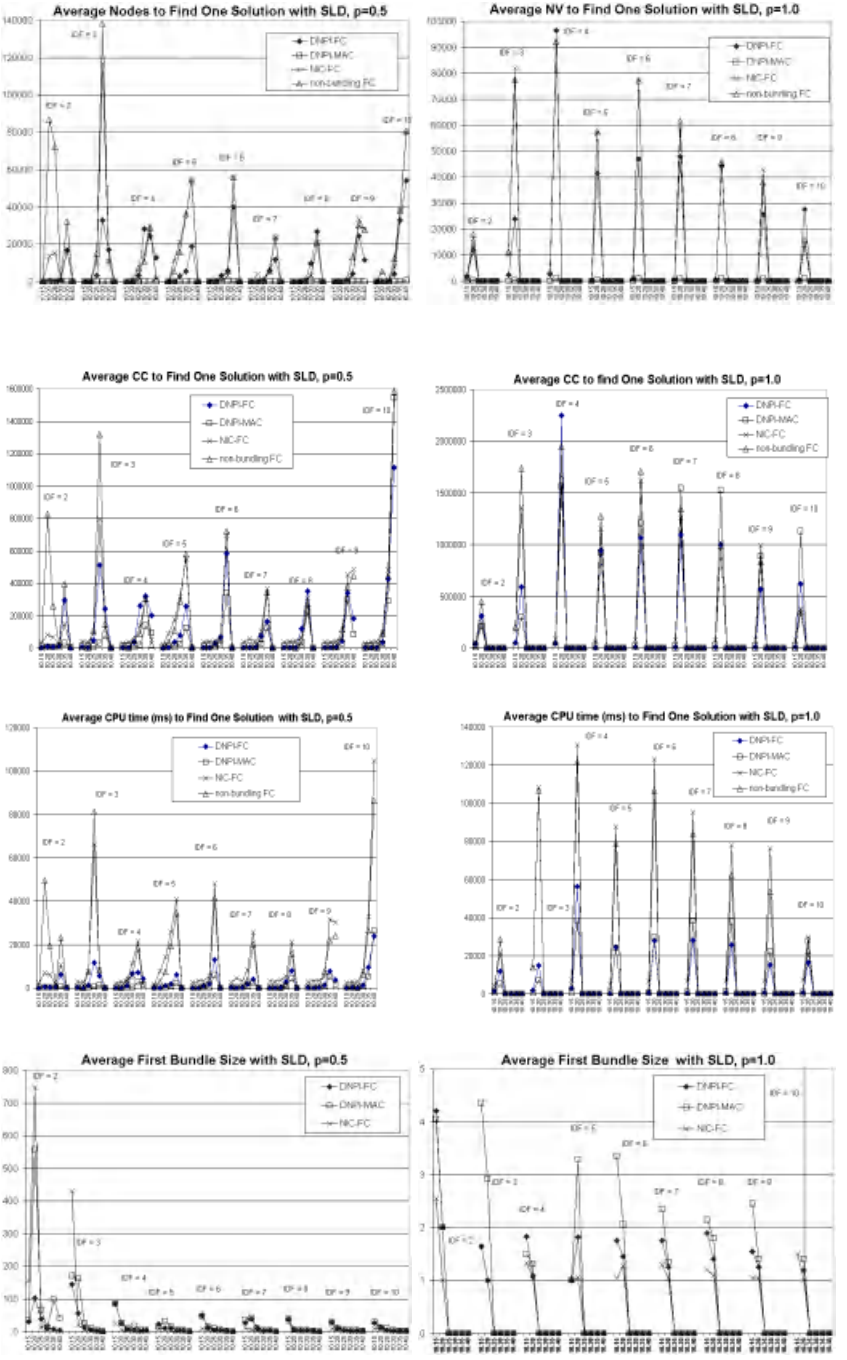



**Fig. 4.** *Bundling with static variable ordering.*

*when finding all solutions*. We see here that MAC expands by far the fewest nodes in the phase transition. It seems to almost not have a phase transition at all. A closer inspection of the data shows that a phase transition is present, even in DNPI-MAC, but is 3 to 4 *orders of magnitude* smaller than the competing methods. DNPI seems to benefit very much from the pairing with MAC in SLD. However, this saving on the number of nodes visited does not extend to the CPU time as noted in Observations 1 and discussed below.

*Constraint checks (CC) with SLD (row 2):* At the left column ($p$=0.5), we notice that, in general, either FC (△) or NIC-FC (×) performs the most constraint checks. In particular, the performance of static bundling (×) is quite disappointing, see IDF= 7, 9 for $p$=0.5 (Observation 7). Moreover, DNPI-MAC performs the fewest constraint checks at every phase transition except when IDF=10. This illustrates an advantage that justifies the use of MAC. Further, dynamic bundling remains the winning strategy as stated in Observation 7.

Recall that the 'traditional' fear of dynamic bundling is the excessive number of constraint checks it requires to compute the interchangeability sets. We show here that, in the most critical region, it is the *non-bundling and static bundling strategies that are actually requiring the most constraint checks*. We are now confident to recommend the use of dynamic bundling in search not only to output several interchangeable solutions (which is useful in practice and the main goal of bundling) but moreover to reduce the severity of the phase transition. This result becomes even more significant under dynamic orderings (see Fig. 5 and 6). In the right column ($p$=1.0), this tendency is less pronounced and all strategies seem to perform fairly similarly: none of them consistently performing fewer or more constraint checks than any other. Nevertheless, the behavior of dynamic bundling never deteriorates the performance of search enough to make it impractical.

*CPU time with SLD (row 3):* The graphs for CPU time bear quite a resemblance with that of constraint checks (Observations 1). However, the distance between the two dynamic bundling strategies and the two others is more clearly visible, in favor of dynamic bundling. Indeed, both dynamic bundling strategies DNPI-FC (⧫) and DNPI-MAC (□) are well below the other strategies in both graphs. This is likely thanks to the significant reduction in the number of nodes visited by these strategies and again supports the use of dynamic bundling to reduce the steepness of the phase transition. Both Observations 7 and 6 are supported here. In the left chart ($p$=0.5), DNPI-MAC (□) usually consumes the least CPU time. In the right chart ($p$=1.0), DNPI-MAC consumes more time than DNPI-FC for high IDF values. This is consistent with the behavior that we observed for constraint checks.

*First bundle size (FBS) with SLD (row 4):* We do not report the First Bundle Size FBS for non-bundling FC, since the solution size is either 1 (when a solution exists) or 0 (when the problem is unsolvable). In general, we find that the sizes of the first bundle found are comparable across strategies (Observation 11) with a few exceptions addressed below. For $p$=0.5 (left), we see that NIC-FC surprisingly performs the best bundling for low values of IDF (i.e., 2 and 3). When IDF

increases to and beyond 4, dynamic bundling regains its advantage: DNPI-FC and DNPI-MAC compete for the larger bundle in most cases. However, for $p$=1.0 (bottom graph), DNPI-MAC nearly always performs the best bundling. Notice that these bundles are quite small, less than 5 solutions are contained in each. One exception is worth mentioning here at the right chart when IDF=10 and $t = 0.15$: the data point here is off the chart indicating an exceedingly large first bundle. This is effect is traced to a *single* instance of the 20 values averaged here and can traced to a weird, but correct, random problem in which 173 out of 190 constraints are identical.

*Conclusions relative to* SLD*:* Under static variable ordering, dynamic bundling, especially when coupled with MAC in DNPI-MAC, drastically reduced the phase transition for a CSP, making the most difficult instances easier to solve.

### 4.3.2 DNPI at Phase Transition: Dynamic Variable Ordering

In general, search strategies with dynamic variable orderings (i.e., DLD and LD-MB) almost always perform better than statically ordered search strategies [2]. In this section we examine DLD and in Section 4.3.3 we will examine LD-MB. The results are presented in Fig. 5. The comparison of Fig. 5 and 4 shows that in general DLD indeed performs better that SLD (the CPU time scale decreases almost tenfold). Moreover, we see that dynamic variable ordering heuristics have a stronger effect on some strategies than others. Specifically, it seems to hurt DNPI-MAC while helping the other strategies. This justifies Observation 4.

*Nodes visited (*NV*) with* DLD *(row 1):* Similar to what we saw for SLD, DNPI-MAC (□) with DLD clearly visits fewer nodes than any other search strategy (Observation 12). Further, we see that the other three strategies DNPI-FC (⧫), NIC-FC (×) and FC (△) compete for visiting the most nodes. DNPI-FC performs the worst most frequently as shown in ($p$=0.5 at IDF = 2, 4, 5, 8 and 9) and ($p$=1.0 at IDF = 2, 4, 8 and 10). Fortunately, poor performance in nodes visited does not otherwise affect the other performance of DNPI-FC, which remains a champion (Observation 16).

*Constraint checks (*CC*) with* DLD *(row 2):* The data here discredits NIC-FC and DNPI-MAC and demonstrates that DNPI-FC is a champion under dynamic ordering. DNPI-MAC performs quite poorly with dynamic ordering, beginning with DLD in Fig. 5 and carrying over LD-MB in Fig. 6. In all cases, DNPI-MAC performs the most constraint checks at the phase transition (Observation 4). In all cases, DNPI-MAC performs the most constraint checks at the phase transition (Observation 4). *Therefore, we safely conclude that the large amount of checks performed by DNPI-MAC is due to the addition of MAC and not to dynamic bundling.* Further, we see that DNPI-FC (⧫) is quite effective. It clearly and significantly reduces the phase transition (Observation 7) and, in general, outperforms the other methods (Observation 16). Interestingly, the strongest competitor to DNPI-FC is FC (△) itself. Note however that FC gives one solution while DNPI-FC gives several robust ones. Neither NIC-FC (Observation 7) nor DNPI-MAC (Observation 4) is worthwhile: they *increase* the phase tran-

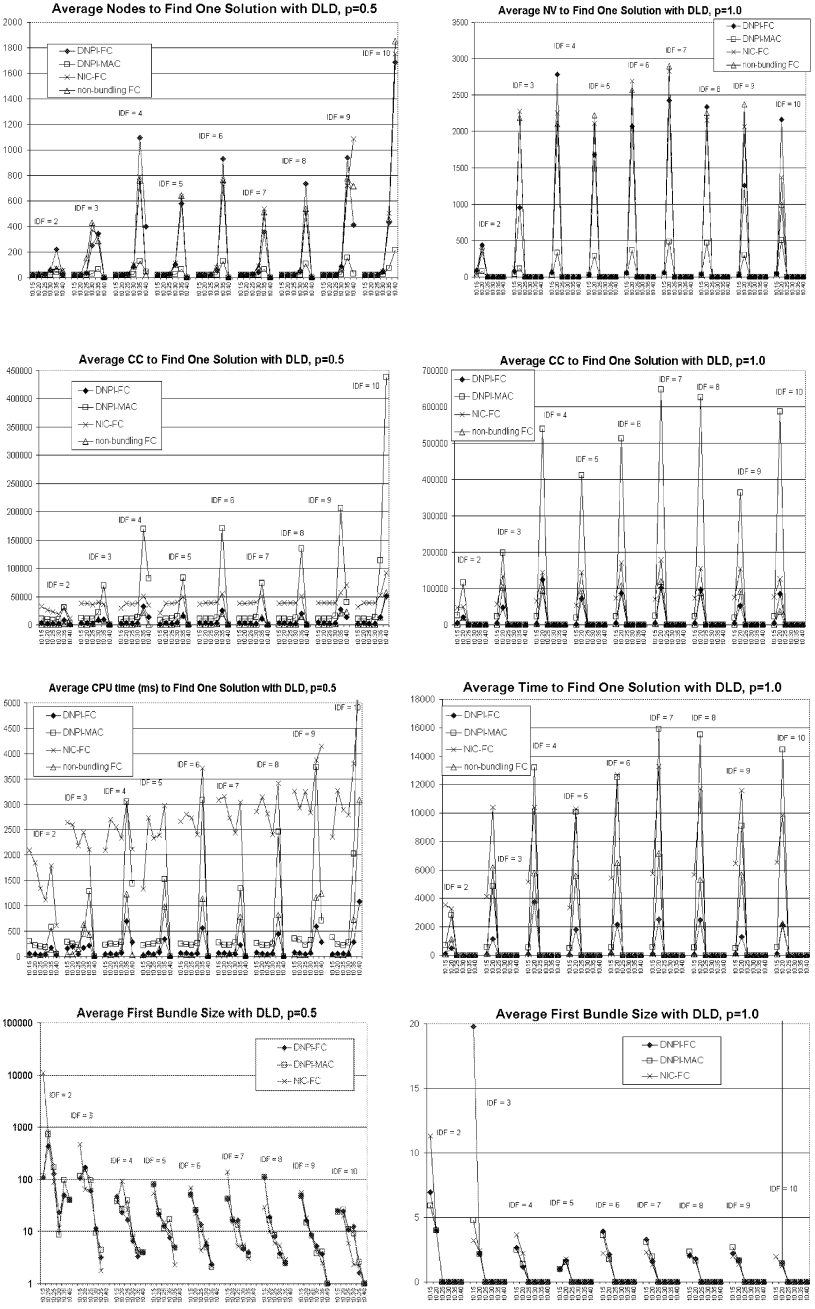



**Fig. 5.** *Bundling with dynamic variable ordering.*

sition rather than decrease it. This justifies our argument in favor of dynamic bundling and confirms our doubts about the appropriateness of MAC in dynamic orderings.

*CPU time with* `DLD` *(row 3):* The charts here amplify the effects discussed above. The disadvantage of static bundling becomes apparent in the left chart, for $p$=0.5 (Observation 7). Though the phase transition is less steep (i.e., the spike is almost concealed), the overall cost of performing NIC-FC (×) search is unnecessarily high. This is due to the overhead of finding all $\mathrm{NI}_C$ interchangeabilities before beginning search. We can also see that DNPI-MAC (□) continues to perform poorly (Observation 4). DNPI-MAC is consistently more costly than DNPI-FC (⧫) and FC (△) even when not at the phase transition. When $p$=1.0, it takes more CPU time than even NIC-FC. Once again, DNPI-FC performs the best overall (Observation 16). In the right graph ($p$=1.0), we see that DNPI-FC (⧫) reduces the phase transition, and performs best at every phase transition (Observations 7 and 16).

*First bundle size (*`FBS`*) with* `DLD` *(row 4):* Finally, we see that though DNPI-MAC (□) puts forth much effort, it does not even produce the best bundles. In reality, NIC-FC (×) performed unexpectedly good bundling, especially where $p$=0.5. DNPI-FC (⧫) also bundles very well; it is even with DNPI-MAC in much of the left charts, and slightly better for most of the right charts (Observation 11).

*Conclusions relative to* `DLD`*:* Under dynamic variable ordering, DNPI-FC continues to perform better than non-bundling FC as bundling effectively reduces the phase transition. Further, the addition of MAC to DNPI is disastrous with a `DLD` ordering: it *increases* the amplitude of the spike of the phase transition (Observation 4). Similarly, NIC-FC behaves worse than FC overall under this ordering (though it finds large bundles).

#### 4.3.3 DNPI at Phase Transition: Dynamic Variable-Value Ordering

The remaining results are shown in Fig. 6. Notice the absence of FC (non-bundling) search on these graphs. Since `LD-MB` is a strategy specific to *bundling*, a non-bundling search strategy such as FC makes no sense in this context. Therefore the comparisons drawn here are between the different bundling strategies. Recall that `LD-MB` is merely `DLD` with a bundle ordering enforced since it assigns to the variable chosen the largest bundle in the partition of its domain. Because of its similarity to `DLD`, it often generally performs as well as `DLD`, but produces a larger first bundle. Revisiting this strategy, we see that its effect on the phase transition (when combined with bundling) is also very similar to `DLD`.

*Nodes visited (*`NV`*) with* `LD-MB` *(row 1):* As for the other orderings, DNPI-MAC (□) visits fewer nodes than any other strategy for both values of $p$ (Observation 12). As with `DLD` in Section 4.3.2, we see that DNPI-FC (⧫) often visits the most nodes. This may serve as a notice that, when looking for only one solution, if it is expensive to expand nodes but cheap to check constraints (here it is the opposite), then DNPI-MAC may be an appropriate choice, as highlighted in Observation 4.

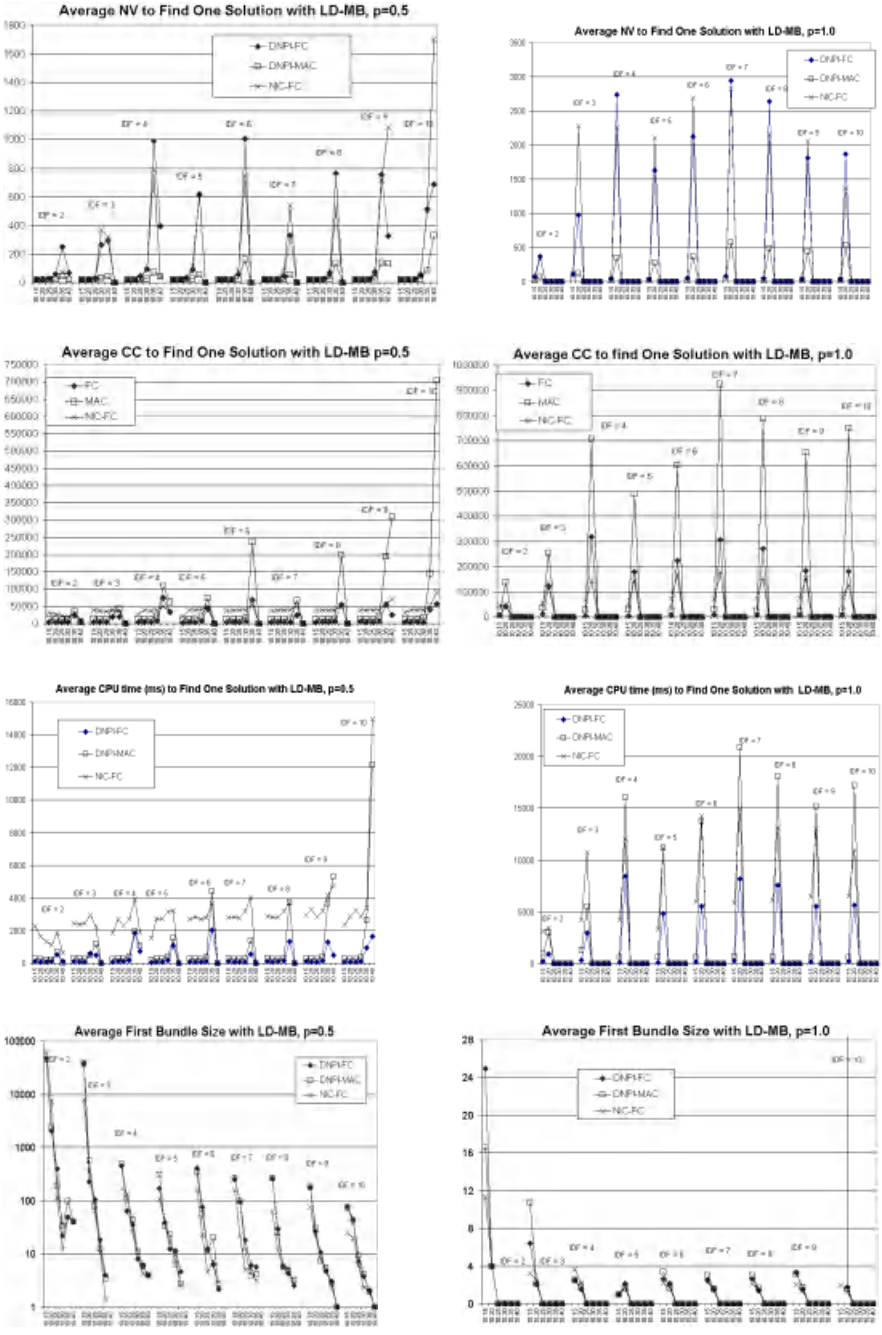



**Fig. 6.** *Bundling with dynamic variable-value ordering.*

*Constraint checks (*CC*) with* LD-MB *(row 2):* Here we see DNPI-MAC (□) checks significantly more constraints than either DNPI-FC (⧫) or NIC-FC (×) at the phase transition (Observation 4). However, notice that at most points, especially for a low $p$ (left graph), NIC-FC (×) performs many more constraint checks than the others, almost concealing the phase transition (Observation 7). This again is due to the overhead of static interchangeability computation. This disadvantage is less visible with a high $p$, where NIC-FC performs more constraint checks for very loose problems ($t = 0.15$), but reduces the phase transition more effectively than either DNPI-FC or DNPI-MAC. The comparison of the second rows of Figs. 5 and 6 shows the following. NIC-FC (×) performs about the same number of constraint checks for LD-MB ordering than for DLD ordering (i.e., between 100,000 and 200,000 when $p$=1.0), but DNPI-FC (⧫) and DNPI-MAC (□) both perform more constraint checks in LD-MB than in DLD (Observation 9). This is a disadvantage of the combination of LD-MB with dynamic bundling. LD-MB requires more backtracking, because the largest bundle in a variable is often a bundle of no-goods, and will trigger backtracking. This will require a re-computation of interchangeability, and becomes more costly in general. Even when looking for only one solution, it seems that DLD is best suited to dynamic bundling. Further, the comparison of second rows in Figs. 4 and 6 confirms an obvious expectation of LD-MB ordering being less expensive and yielding better bundles than static variable ordering SLD (Observation 10).

*CPU time with* DLD *(row 3):* The trends we noted in the nodes visited and constraints checked for LD-MB consistently extend to the CPU time consumed. We see that both NIC-FC (×) and DNPI-MAC (□) have generally poor performance, requiring more time that DNPI-FC (⧫). This confirms Observations 7, 4, and 16. Also, the comparison of CPU time (third row) with the first two ordering heuristics (i.e., SLD in Fig. 4 and DLD in Fig. 5) confirm Observations 10 and 9, respectively.

*First bundle size (*FBS*) with* LD-MB *(row 4):* We see that, based on the size of the first bundle, no particular bundling strategy can be declared a *clear* winner. In general, dynamic bundling is a little stronger than static bundling, with two exceptions ($p$=0.5, IDF=2 and $p$=1.0, IDF=4). As far as lookahead strategies are concerned, DNPI-MAC and DNPI-FC are comparable and quite competitive with respect to their bundling capabilities, thus justifying again the power of the dynamic bundling and the superfluity of MAC (Observation 11). The comparison across ordering heuristics show that LD-MB yields better bundles than both SLD (Observation 10) and DLD (Observation 9)

*Conclusions relative to* LD-MB*:* It appears clearly that dynamic bundling *without* MAC, that is DNPI-FC, effectively reduces the phase transition and produces large bundles across all variable ordering heuristics. Further, we note that, in the phase transition, DLD seems to be the most effective ordering.

## 5 Conclusions

Regarding the lookahead strategy to use with dynamic bundling, we establish that, although MAC visits fewer nodes than FC, it requires in general more constraint checks and more CPU time. This is especially true in dynamic variable ordering (i.e., `DLD` and `LD-MB`), where the cost of MAC becomes a serious impediment. In conclusion, unless we are using `SLD`, DNPI-MAC is not worth the effort and DNPI-FC should be used instead.

Regarding the phase transition, we prove that dynamic bundling is uniformly worthwhile by reducing the spike at the cross-over point. This is especially true for DNPI-FC. DNPI-MAC is a good search strategy to reduce the phase transition if static ordering must be used, but if dynamic ordering is permitted, DNPI-FC is much more effective. As a conclusion, the phase transition is best reduced by DNPI-FC with `DLD` variable ordering, which has never before been implemented. These strategies, and their relative rank (1st, 2nd and 3rd) are summarized in Figure 7.

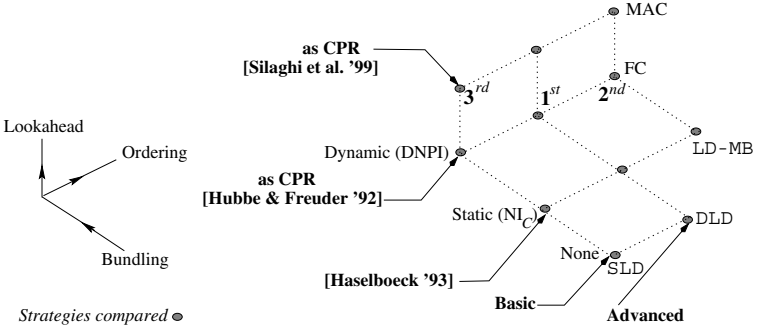



**Fig. 7.** *A summary of the strategies implemented and tested and the best ranking ones. All strategies not otherwise marked were proposed by us.*

## References

1. Amy M. Beckwith and Berthe Y. Choueiry. On the Dynamic Detection of Interchangeability in Finite Constraint Satisfaction Problems. In T. Walsh, editor, $7^{th}$ *International Conference on Principle and Practice of Constraint Programming (CP'01)*, LNCS Vol. 2239, page 760, Paphos, Cyprus, 2001. Springer Verlag.
2. Amy M. Beckwith, Berthe Y. Choueiry, and Hui Zou. How the Level of Interchangeability Embedded in a Finite Constraint Satisfaction Problem Affects the Performance of Search. In *14th Australian Joint Conference on Artificial Intelligence. LNAI Vol. 2256*, pages 50–61, Adelaide, Australia, 2001. Springer Verlag.
3. Cynthia A. Brown, Larry Finkelstein, and Paul W. Purdom, Jr. Backtrack Searching in the Presence of Symmetry. In T. Mora, editor, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pages 99–110. Springer-Verlag, 1988.

4. Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the Really Hard Problems Are. In *Proc. of the 12 th IJCAI*, pages 331–337, Sidney, Australia, 1991.
5. Berthe Y. Choueiry and Amy M. Beckwith. On Finding the First Solution Bundle in Finite Constraint Satisfaction Problems. Technical Report CSL-01-03. `consystlab.unl.edu/CSL-01-04.ps`, University of Nebraska-Lincoln, 2001.
6. Berthe Y. Choueiry, Boi Faltings, and Rainer Weigel. Abstraction by Interchangeability in Resource Allocation. In *Proc. of the 14 th IJCAI*, pages 1694–1701, Montréal, Québec, Canada, 1995.
7. Berthe Y. Choueiry and Guevara Noubir. On the Computation of Local Interchangeability in Discrete Constraint Satisfaction Problems. Technical Report KSL-98-24, Knowledge Systems Laboratory, Department of Computer Science, Stanford University, Stanford, CA, 1998. Preliminary version in *Proc. of AAAI'98*.
8. Thomas Ellman. Abstraction via Approximate Symmetry. In *Proc. of the 13 th IJCAI*, pages 916–921, Chambéry, France, 1993.
9. Jay P. Fillmore and S.G. Williamson. On Backtracking: A Combinatorial Description of the Algorithm. *SIAM Journal on Computing*, 3 (1):41–55, 1974.
10. Eugene C. Freuder. Eliminating Interchangeable Values in Constraint Satisfaction Problems. In *Proc. of AAAI-91*, pages 227–233, Anaheim, CA, 1991.
11. Ian P. Gent and Patrick Prosser. Inside MAC and FC. Technical Report APES-20-2000, APES Research Group, 2000.
12. Matthew L. Ginsberg, Andrew J. Parkes, and Amitabha Roy. Supermodels and Robustness. In *Proc. of AAAI-98*, pages 334–339, Madison, Wisconsin, 1998.
13. J.W.L. Glaisher. On the Problem of the Eight Queens. *Philosophical Magazine, series 4*, 48:457–467, 1874.
14. Alois Haselböck. Exploiting Interchangeabilities in Constraint Satisfaction Problems. In *Proc. of the 13 th IJCAI*, pages 282–287, Chambéry, France, 1993.
15. Tad Hogg, Bernardo A. Hubermann, and Colin P. Williams, editors. *Special Volume on Frontiers in Problem Solving: Phase Transitions and Complexity*, volume 81 (1-2). Elsevier Science, 1996.
16. Jean-Francois Puget. On the Satisfiability of Symmetrical Constrained Satisfaction Problems. In *ISMIS'93*, pages 350–361, 1993.
17. Daniel Sabin and Eugene C. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proc. of the 11 th ECAI*, pages 125–129, Amsterdam, The Netherlands, 1994.
18. M-C. Silaghi, D. Sam-Haroud, and B. Faltings. Ways of Maintaining Arc Consistency in Search using the Cartesian Representation. In *Proc. of ERCIM'99*, LNAI, Paphos, Cyprus, 1999. Springer Verlag.

# Symbolic Heuristic Search Using Decision Diagrams

Eric Hansen[1], Rong Zhou[1], and Zhengzhu Feng[2]

[1] Computer Science Department, Mississippi State University, Mississippi State MS
{hansen,rzhou}@cs.msstate.edu,
[2] Computer Science Department, University of Massachusetts, Amherst MA
fengzz@cs.umass.edu

**Abstract.** We show how to use symbolic model-checking techniques in heuristic search algorithms for both deterministic and decision-theoretic planning problems. A symbolic approach exploits state abstraction by using decision diagrams to compactly represent sets of states and operators on sets of states. In earlier work, symbolic model-checking techniques have been used to find plans that minimize the number of steps needed to reach a goal. Our approach generalizes this by showing how to find plans that minimize the expected cost of reaching a goal. For this generalization, we use algebraic decision diagrams instead of binary decision diagrams. In particular, we show that algebraic decision diagrams provide a compact representation of state evaluation functions. We describe symbolic generalizations of A* search for deterministic planning and of LAO* search for decision-theoretic planning problems formalized as Markov decision processes. We report experimental results and discuss issues for future work.

## 1 Introduction

There is currently great interest in using symbolic model-checking to solve AI planning problems with large state spaces [1,2,3,4,5,6,7,8]. This interest is based on recognition that the problem of finding a plan (i.e., a sequence of actions and states leading from an initial state to a goal state) can be treated as a reachability problem in model checking. The planning problem is solved symbolically in this sense: reachability analysis is performed by manipulating sets of states, rather than individual states. Sets of states (and operators on sets of states) are represented compactly using *decision diagrams*. This approach exploits state abstraction to solve problems with large state spaces. For problems with regular structure, decision diagrams can often provide a polynomial representation of an exponential number of states. A symbolic approach to model-checking has made it possible to verify systems with more than $10^{30}$ states, and a similar approach to scaling up planning algorithms shows much promise.

Symbolic model-checking techniques have been explored for both deterministic and nondeterministic planning. However, the algorithms that have been developed only find plans that are "optimal" in the sense that they minimize the

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 83–98, 2002.

number of actions needed to reach a goal. The more general problem of finding minimal-cost plans, where there is a varying or state-dependent cost for actions, has not yet been addressed (or has not been addressed adequately). We argue that this follows, in part, from the use of binary decision diagrams in previous work. We adopt algebraic decision diagrams as a more general data structure for symbolic heuristic search. Whereas binary decision diagrams represent Boolean functions, algebraic decision diagrams can represent arbitrary real-valued functions. We show that this allows us to develop algorithms for deterministic and decision-theoretic planning that can find minimal-cost plans.

The paper is organized as follows. In section 2, we review the framework of state-space heuristic search and present some relevant background for a symbolic approach to heuristic search. We review binary decision diagrams and algebraic decision diagrams. We also review previous work in which decision diagrams are used to support state abstraction in AI planning. In section 3, we describe a symbolic generalization of A* that can be used to solve deterministic planning problems. In section 4, we describe a symbolic generalization of LAO*, a heuristic search algorithm for solving decision-theoretic planning problems formalized as Markov decision processes. We present experimental results for both algorithms. We conclude the paper with a comparison of the two algorithms and a discussion of issues for future work.

## 2 Background

### 2.1 State-Space Search

We begin by reviewing the framework of state-space search and introduce some notation that will be used in the rest of the paper.

A state-space search problem is defined as a tuple $(S, G, s^0, A, T, c)$, where $S$ denotes a set of states; $G \subset S$ denotes a set of goal states; $s^0$ denotes a start state; $A$ denotes a set of actions (or operators); $T$ denotes a set of transition models $\{T^a\}$, one for each action $a \in A$, describing the state transitions caused by each action; and $c$ denotes a set of cost models $\{c^a\}$, one for each action, specifying the state-dependent (expected) cost of actions.

We consider two kinds of transition model in this paper. In a deterministic transition model, an action $a \in A$ in state $s \in S$ always leads to the same successor state $s' \in S$. We can represent this by a function $T^a : S \times S \to \{0, 1\}$, where the value of the function is 1 when action $a$ causes a transition from state $sS$ to state $s'$. In a stochastic transition model, an action $a \in A$ in state $s \in S$ has several possible successor states, each occurring with some probability. We can represent this by a function $T^a : S \times S \to [0, 1]$, where the value of the function is the probability that action $a$ causes a transition from state $s$ to state $s'$. The first kind of state-space search problem is a deterministic shortest-path problem. The second is a special kind of Markov decision process called a stochastic shortest-path problem [9]. Both can be solved by dynamic programming. But given a start state $s^0 \in S$ and an admissible heuristic function $h : S \to \Re$, an optimal solution

can also be found by a heuristic search algorithm that gradually expands a partial solution, beginning from the start state and guided by the heuristic, until a complete solution is found. The heuristic search approach has the advantage that it only needs to evaluate part of the state space to find an optimal solution. We consider two heuristic search algorithms in this paper; the A* algorithm for deterministic planning problems [10] and the LAO* algorithm for stochastic planning problems [11]. The contribution of this paper is to describe how both of these algorithms can exploit state abstraction using decision diagrams.

### 2.2 Factored State Representation

To use decision diagrams for state abstraction, we adopt a binary encoding of states. We assume the set of states $S$ is described by a set of Boolean variables $\mathbf{X} = \{X_1, \ldots, X_n\}$, where $\mathbf{x} = \{x_1, \ldots, x_n\}$ denotes a particular instantiation of the variables, corresponding to a unique state $s \in S$. In the planning literature, such Boolean variables are called *fluents*. In the literature on decision-theoretic planning, a state space structured in this way is called a *factored* Markov decision process. Thus we represent the transition and cost models of a state-space search problem as a mapping defined over a Boolean encoding of the state space. For example, we represent the transition model of a planning problem by the function $T^a(\mathbf{X}, \mathbf{X}')$, where $\mathbf{X} = \{X_1, \ldots, X_n\}$ refers to the set of state variables before taking action $a$ and $\mathbf{X}' = \{X'_1, \ldots, X'_n\}$ refers to the set of state variables after taking the action. Similarly, we represent the cost model by the function $c^a(\mathbf{X})$. Although the set of states $S = 2^{\mathbf{X}}$ grows exponentially with the number of variables, decision diagrams can exploit state abstraction to represent these models compactly.

### 2.3 Decision Diagrams

The decision diagram data structure was developed by the symbolic model-checking community for application to VLSI design and verification [12,13]. A decision diagram is a compact representation of a mapping from a set of Boolean state variables to a set of values. A binary decision diagram (BDD) represents a mapping to the values 0 or 1. An algebraic decision diagram (ADD) represents a more general mapping to any discrete set of values (including real numbers) [14].

A decision diagram is a directed acyclic graph with internal nodes of out-degree two (corresponding to Boolean variables) and a terminal node for each value of the function it represents. In an ordered decision diagram, the Boolean variables in all paths through the graph respect a linear order $X_1 \prec X_2 \prec \cdots \prec X_n$. An ordered decision diagram can be reduced by successive applications of two rules. The *deletion rule* eliminates a node with both of its outgoing edges leading to the same successor. The *merging rule* eliminates one of two nodes with the same label as well as the same pair of successors. A reduced decision diagram with a fixed variable ordering is a canonical representation of a function. The canonicity of decision diagrams gives rise to the existence of efficient algorithms for manipulating them. Because the complexity of operators on decision diagrams

depends on the number of nodes in the diagrams and not on the size of the state space, decision diagrams can exploit regularity in the model to solve problems with large state spaces more efficiently.

A symbolic approach to both model checking and heuristic search manipulates sets of states, instead of individual states. A set of states $S$ is represented by its characteristic function $\chi_S$, so that $s \in S \iff \chi_S(\mathbf{X}) = 1$. The characteristic function is represented compactly by a BDD. (From now on, whenever we refer to a set of states, $S$, we implicitly refer to its characteristic function, represented by a decision diagram.) In both model checking and heuristic search, a central task is to determine the set of states that is reachable from an initial state. This can be achieved through a simple breadth-first search starting from the initial state, until no more new states can be added. Let $S^i$ denote the set of states reachable from the initial state $s^0$ in $i$ steps, initialized by $S^i = \{s^0\}$. There is an efficient algorithm for determining the next step characteristic functions $\chi_{S^{i+1}}(\mathbf{X})$ given the current characteristic function $\chi_{S^i}(\mathbf{X})$. Recall that $T(\mathbf{X}, \mathbf{X}')$ is true if and only if $\mathbf{X}$ is the encoding of a given state and $\mathbf{X}'$ is the encoding of its successor state. The next-step characteristic function is computed as follows:

$$\chi_{S^{i+1}}(\mathbf{X}') = \exists_{X \in \mathbf{X}}(\chi_{S^i}(\mathbf{X}) \wedge T(\mathbf{X}, \mathbf{X}')) \tag{1}$$

Equation (1) is called the *relational product* operator. It represents a series of nested existential quantifications, one for each variable in $\mathbf{X}$. The conjunction $T^a(\mathbf{X}, X') \wedge \chi_S(\mathbf{X})$ selects the set of valid transitions and the existential quantification extracts and unions the successor states together. Given $\chi_{S^0}(\mathbf{X})$, we can iteratively compute the characteristic function $\chi_{S^1}(\mathbf{X}), \chi_{S^2}(\mathbf{X}), \chi_{S^3}(\mathbf{X}), \cdots$, until the transitive closure of the transition relation is computed. Both the relational-product operator and symbolic traversal algorithms are well studied in the symbolic model checking literature, and we refer to that literature for further details [15].

### 2.4 Deterministic Planning

We are not the first to explore the use of decision diagrams in heuristic search. Edelkamp and Reffel [3] describe a symbolic generalization of A*, called BDDA*, that can solve deterministic planning problems. It uses BDDs to represent sets of states and operators on sets of states in a compressed form that exploits structure in the problem domain. They show that symbolic search guided by a heuristic significantly outperforms breadth-first symbolic search. They also show that this approach is effective in solving a class of deterministic planning problems [4].

In Section 3, we describe an alternative implementation of symbolic A* that uses ADDs. Because BDDA* uses BDDs as a data structure, the cost-functions $c^a$, the state evaluation function $f$, the heuristic evaluation function $h$, and the open list, must all be encoded in binary. Although it is possible to encode anything in binary, this makes the implementation of BDDA* awkward in several respects. In fact, the original version of BDDA* could only solve problems for which all actions have unit cost and the objective is to find the *shortest* plan.

Edelkamp [16] later described a more general implementation of BDDA* does not assume unit-cost actions. However, we argue that ADDs provide a more natural representation for such problems.

### 2.5 Nondeterministic and Decision-Theoretic Planning

Decision diagrams have also been used for nondeterministic and decision-theoretic planning. Nondeterministic planning considers actions with multiple outcomes and allows plan execution to include conditional and iterative behavior. However, it does not associate probabilities and costs with state transitions, as in decision-theoretic planning. In decision-theoretic planning, the problem is not simply to construct a plan that can reach the goal, but to find a plan that minimizes expected cost (or equivalently, maximizes expected value).

There has been substantial work on using BDDs for nondeterministic planning [1,2]. There has also been work on using ADDs to exploit state abstraction in dynamic programming for MDPs [7,8]. In Section 4, we show how to integrate these two approaches to create a symbolic heuristic search algorithm that can solve decision-theoretic planning problems formalized as MDPs.

## 3 Deterministic Planning: Symbolic A*

We first describe an implementation of A* that uses ADDs instead of BDDs as a data structure. We call this algorithm ADDA*, in contrast to Edelkamp and Reffel's BDDA* algorithm. We show that this change of data structure leads to a simpler and more natural implementation of symbolic A* that can solve cost-minimization problems.

### 3.1 Algorithm

In the preceding section, we reviewed how symbolic model-checking techniques are used to compute the set of states reachable from a start state. For reachability analysis, it is sufficient to compute successor states. But for a heuristic search algorithm such as A*, we also need to compute and store state costs. We begin by pointing out that an algebraic decision diagram provides a symbolic representation of a real-valued function, and thus a more natural representation for the cost function, $c$, heuristic function $h$, and state evaluation function $f$, of A*. No binary encoding of these real-valued functions is necessary, as in BDDA*. As we will show, this simplifies our implementation of A*, without sacrificing the advantages of a symbolic representation.

Recall that A* stores the search frontier in a list called $OPEN$, which organizes nodes in increasing order of their $f$-cost. The node with the least $f$-cost is selected to generate all its successor nodes, a process known as a node expansion. Note that the $OPEN$ list for A* can be viewed as a function that maps states to costs. If the $f$-cost of many nodes are the same, then a considerable degree of state abstraction can be achieved by treating states with the same $f$-cost as

an aggregate or abstract state. This is the approach adopted by BDDA* and it is the same approach we adopt for ADDA*. It makes it possible to perform node expansion symbolically by expanding a set of states (with the same $f$-cost), instead of a single state at a time.

BDDA* represents the $OPEN$ list as a set of BDDs, one for each distinct $f$-cost. Each BDD represents the characteristic function of a set of states with the same $f$-cost. By contrast, ADDA* represents the open list as a single ADD, where the leaves of the ADD correspond to the distinct $f$-costs in the open list. This representation is more natural, and as we will show, it makes it easier to implement the rest of the A* algorithm.

When we expand a set of states in A*, we determine their successor states and compute their $g$ and $f$-costs. In graph search, we also need to detect states that represent duplicate paths and preserve the one with the least $g$-cost. Edelkamp and Reffel's original implementation of A* determines the set of successor states separately from computing their $g$ and $f$-costs, and does not detect duplicate nodes.[1] Using ADDs instead of BDDs as a data structure makes it possible to do all of these things at the same time.

But to do so, we need to generalize the relational product operator defined for BDDs so that it also works with ADDs. The standard operator cannot be used with ADDs because the terminals of ADDs correspond to real values, and not necessarily the Boolean constants $\{0, 1\}$. In Boolean algebra, the relational product is computed by using existential quantification given by the following formula:

$$\exists_X f(\mathbf{X}) = f(\mathbf{X})|_{X=0} \vee f(\mathbf{X})|_{X=1} \tag{2}$$

A naive way of adapting existential quantification to ADDs is to substitute the algebraic addition operator $(+)$ for the Boolean disjunction operator $(\vee)$ in equation (2). However, this won't generalize to the case of graph search. In graph search, there may be more than one path from the start state to a given state. Only one of these paths should be preserved and the minimal-cost path is preferred. But if the naive approach to adapting existential quantification to ADDs is used, the cost of a successor state would be the sum of the costs of all duplicate paths to that state! Because this does not make sense, we invented a new operator called *existential least-cost quantification*. It is defined by the following equation:

$$\exists_X^{LC} f(\mathbf{X}) = \min(f(\mathbf{X})|_{X=0}, f(\mathbf{X})|_{X=1}) \tag{3}$$

The intuitive meaning of the existential least-cost quantification operator is to preserve only the path that has the least cost to a given state, discarding any suboptimal paths.

In order to make existential least-cost quantification work seamlessly with the transition function $T$, we change the definition of the latter as follows: $T(\mathbf{X}, \mathbf{X}')$

[1] Edelkamp [16] describes a later implementation of BDDA * that does detect duplicate nodes using a technique called *forward set simplification*, although this technique can only detect duplicate paths to nodes that are already expanded.

**Table 1.** Pseudocode for ADDA*

**procedure** ADDA* $(\chi_{S^0}, \chi_{S^g})$
1. $OPEN(\mathbf{X}, f) \leftarrow (\chi_{S^0}, h(\chi_{S^0})) \wedge (\neg\chi_{S^0}, \infty)$
2. **while** $OPEN \not= \emptyset$ **do**
3. $\quad \mathcal{B} \leftarrow \{\beta | (\beta, n \in OPEN) \wedge (f(\beta) = \min f(n))\}$
4. $\quad$ **if** $\exists_{X \in \mathbf{X}}(\mathbf{X}(\mathcal{B}) \wedge \chi_{S^g})$ **then return** $f(\mathcal{B})$
5. $\quad OPEN^- \leftarrow OPEN \setminus \mathcal{B}$
6. $\quad \mathcal{G}(\mathbf{X}, g) \leftarrow (\mathbf{X}(\mathcal{B}), f(\mathcal{B}) - h(\mathcal{B}))$
7. $\quad \mathcal{G}(\mathbf{X}', g) \leftarrow \exists^{LC}_{X \in \mathbf{X}} \bigcup_a (\mathbf{X}(\mathcal{G}) * T^a(\mathbf{X}, \mathbf{X}') + c^a(\mathbf{X}))$
8. $\quad \mathbf{X}(\mathcal{G}) \leftarrow SwapVar_{\mathbf{X}', \mathbf{X}}(\mathbf{X}'(\mathcal{G}))$
9. $\quad OPEN^+ \leftarrow (\mathbf{X}(\mathcal{G}), g(\mathcal{G}) + h(\mathbf{X}(\mathcal{G})))$
10. $\quad OPEN \leftarrow \min(OPEN^-, OPEN^+)$

maps to constant 1 if and only if $\mathbf{X}$ is the encoding of a given state and $\mathbf{X}'$ is the encoding of its successor state; otherwise, it maps to an infinity ($\infty$) constant. This modification of the transition function $T$ is necessary in order to make the cost of any illegal move infinity. As a consequence, all illegal moves will be ignored.

In order to detect duplicate paths in graph search and preserve the one with the least $g$-cost, ADDA* performs existential least-cost quantification on an ADD that represents the $g$-costs of the set of states being expanded. First it selects the set of states to expand by selecting those with the least $f$-cost. ADDA* computes the ADD representing the $g$-cost of the states by subtracting the heuristic estimate from their $f$-costs, as follows: $g(\mathbf{X}) = f(\mathbf{X}) - h(\mathbf{X})$. Then ADDA* uses least-cost existential quantification to compute the successors of this set of states and their $g$-costs. The $f$-costs of the successor states after taking action $a$ are computed as follows:

$$\begin{aligned} f^a(\mathbf{X}') &= g(\mathbf{X}') + h(\mathbf{X}') \\ &= g(\mathbf{X}) + c^a(\mathbf{X}) + h(\mathbf{X}') \\ &= f(\mathbf{X}) - h(\mathbf{X}) + c^a(\mathbf{X}) + h(\mathbf{X}'), \end{aligned}$$

and the successor states for each action are inserted into the open list.

Figure 1 shows the pseudocode of ADDA*. Let $\chi_{S^0}, \chi_{S^g}$ denote the characteristic function of the starting node(s) and goal node(s), respectively. Initially, $OPEN$ contains a single starting node with its $f$ value set to the heuristic estimate $h$ and the $f$ value for everything else is set to infinity. Then the algorithm finds the set of nodes with the least $f$ value, $\mathcal{B}$. Once the intersection between the set $\mathcal{B}$ and the goal set $\chi_{S^g}$ is not the trivial zero function, this indicates a goal node has been found. If no goal node is found, the set $\mathcal{B}$ is detached from the $OPEN$ and the $g$ values of node $\in \mathcal{B}$ are computed and stored in $\mathcal{G}$. Since the transition function $T$ has a value of 1 for any legal move, which means after the product of "$\mathbf{X}(\mathcal{G})$" and "$T(\mathbf{X}, \mathbf{X}')$" in line# 7, the $g$ value of all legal moves

will be preserved, while the $g$ value of all illegal moves will be set to infinity and discarded later. Furthermore, according to the definition of existential least-cost quantification, among all the legal moves, only the move that produces the least-cost path will be kept. Which means for the successors of the nodes with the minimum $f$ value, only the best (shortest) paths are stored in the $OPEN$. The last line (line# 10) adds the successor nodes back to the original $OPEN$. The purpose of using a "min" operator is to check for duplicate paths between the set of newly generated successors nodes and the set of old nodes in $OPEN$. We note that the data structure we refer to as $OPEN$ is the union of the open and closed lists of A* and contains all states evaluated by A*. Although the open and closed lists can be represented by distinct data structures in symbolic A*, we found that it is more memory-efficient to represent them by a single ADD.

### 3.2 Experimental Results

We implemented ADDA* and compared its performance to a symbolic breadth-first search algorithm implemented using BDDs, called BDD-BFS.[2] We tested both algorithms on 200 randomly generated instances of the Eight Puzzle and the Fifteen Puzzle. (For the Fifteen Puzzle, we only considered instances with solution depth up to 40.) As shown in Table 2, ADDA* stores about one fifth as many nodes as BDD-BFS and is more than twice as fast in solving the Eight Puzzle. None of the Fifteen Puzzle instances could be solved by BDD-BFS. The performance of ADDA* on the Fifteen puzzle is shown in the table. We note that these results are consistent with the results reported by Edelkamp and Reffel for BDDA* in solving the Eight and Fifteen Puzzles. Our current algorithm does not seem to be more or less efficient than theirs. However, it is simpler and more general, by virtue of being implemented using ADDs instead of BDDs.

**Table 2.** Performance comparison of ADDA* and breadth-first search (BDDBFS).

| Problem | Size | | Time | |
|---|---|---|---|---|
| | BDDBFS | ADDA* | BDDBFS | ADDA* |
| 8-puzzle | 28096 | 5465 | 8.91 | 3.89 |
| 15-puzzle | Unsolvable | 548546 | Unsolvable | 902.85 |

We also compared ADDA* and ordinary A* in solving the Fifteen Puzzle. On average, A* generates 434,282 nodes. Although this seems less than the number of nodes generated by ADDA* (548, 546), a node in ADDA* is not the same as a node in A*. A decision diagram node can only represent a single bit (true or false) in the encoding of a Fifteen Puzzle state; whereas a node in A* can

[2] The decision diagram package used to implement our algorithm is the CUDD package from University of Colorado [17]. We modified the package to support the existential least-cost quantification operator. Experiments were performed on a Sun UltraSPARC II with a 300MHz processor and 2 gigabytes of memory.

represent the entire state of the board. In our implementations, a node in A* takes roughly two and a half times the physical memory required for a node in ADDA* and we found that the amount of physical memory saved using ADDA* instead of ordinary A* is slightly more than 50%. This reflects the benefit of state abstraction. In solving sliding-tile puzzles, however, there is not sufficient state abstraction to compensate for the significant overhead involved in using decision diagrams and ordinary A* is faster than ADDA*.

Our experimental results for ADDA* are very preliminary. There are several ways in which its performance may be improved, and we plan to test it on a range of different problems. We discuss some of the possibilities for future work in the conclusion of the paper.

## 4 Decision-Theoretic Planning: Symbolic LAO*

We now turn to a more complex class of planning problems in which state transitions are stochastic, and, as a result, plans have a more complex structure that includes branches and loops. Previous authors have described how to use symbolic reachability analysis to solve nondeterministic planning problems. But nondeterministic planning does not consider the probability of state transitions or their cost. Hoey et al. [7] have described a symbolic dynamic programming algorithm for Markov decision processes. It does consider the probability of state transitions and their cost, and uses ADDs to aggregate states with the same value – the same approach to state abstraction we used for A*. However, their SPUDD planner is a dynamic programming algorithm that solves the problem for all possible starting states, without considering reachability. In the rest of this paper, we describe a heuristic search approach to solving this class of problems that integrates both approaches – reachability analysis and dynamic programming.

### 4.1 Algorithm

Our algorithm is a symbolic generalization of the LAO* algorithm, a heuristic search algorithm for stochastic shortest-path problems (and other Markov decision problems) [11]. Given a start state, LAO* finds an optimal policy that can include cycles and reaches the goal state with probability one. It is interesting to note that a policy that reaches the goal state with probability one can be viewed as a decision-theoretic generalization of the concept of a *strong cyclic plan* in nondeterministic planning [18].

LAO* is an extension of the classic search algorithm AO* [19]. Like AO*, it has two alternating phases. First, it expands the best partial solution (or policy) and evaluates the states on its fringe using a heuristic evaluation function. Then it performs dynamic programming on the states visited by the best partial solution, to update their values and possibly revise the best current solution. The two phases alternate until a complete solution is found, which is guaranteed to be optimal. LAO* differs from AO* by allowing solutions with loops, requiring it to use a more general dynamic programming algorithm such as value iteration or

**Table 3.** Symbolic LAO* algorithm.

**procedure** solutionExpansion($\pi, \chi_{S^0}, G$)
1. $E \leftarrow F \leftarrow \phi$
2. $from \leftarrow \chi_{S^0}$
3. **repeat**
4. $\quad to \leftarrow \bigcup_a \exists_{X \in \mathbf{x}}((from \cap \chi_{S^a_\pi}) \wedge T^a(\mathbf{X}, \mathbf{X}'))$
5. $\quad F \leftarrow F \cup\ (to - G)$
6. $\quad E \leftarrow E \cup from$
7. $\quad from \leftarrow to \cap G - E$
8. **until** $(from = \phi)$
9. $E \leftarrow E \cup F$
10. $G \leftarrow G \cup F$
11. **return** $(E, F, G)$

**procedure** dynamicProgramming($E, f$)
12. $savef \leftarrow f$
13. $E' \leftarrow \bigcup_a \exists_{X \in \mathbf{x}}(E \wedge T^a(\mathbf{X}, \mathbf{X}'))$
14. **repeat**
15. $\quad f' \leftarrow f$
16. $\quad$ FOR each action $a$
17. $\quad\quad f^a \leftarrow c^a_E + \sum_{E'} T^a_{E \cup E'} f'_{E'}$
18. $\quad M \leftarrow \min_a f^a$
19. $\quad f \leftarrow M \cup savef_{\overline{E}}$
20. $\quad$ residual $\leftarrow \|f_E - f'_E\|$
21. **until** stopping criterion met
22. $\pi \leftarrow extractPolicy(M, \{f^a\})$
23. **return** $(f, \pi, residual)$

**procedure** LAO*($\{T^a\}, \{c^a\}, \chi_{S^0}, h, threshold$ )
24. $f \leftarrow h$
25. $G \leftarrow \phi$
26. $\pi \leftarrow 0$
27. **repeat**
28. $\quad (E, F, G) \leftarrow solutionExpansion(\pi, \chi_{S^0}, G)$
29. $\quad (f, \pi, residual) \leftarrow dynamicProgramming(E, f)$
30. **until** $(F = \phi)$ **and** (residual $\leq$ threshold)
31. **return** $(\pi, f, E, G)$.

policy iteration. To create a symbolic generalization of LAO*, we will implement the first phase of the algorithm using a variant of symbolic reachability analysis and the second phase using a variant of the SPUDD algorithm.

To integrate the two phases of LAO* – reachability analysis and dynamic programming – we introduce the concept of *masking*. To motivate this idea, we note that all elements manipulated by our symbolic generalization of LAO* are represented by ADDs (including the transition and cost models, the policy $\pi : S \rightarrow A$, the state evaluation function $f : S \rightarrow \Re$,the admissible heuristic $h : S \rightarrow$

$\Re$, etc.), and all computations are performed using ADDs. A potential problem is that an ADD assigns a value to every state in the state space. However, we want to limit computation to the set of states that are reachable from the start state by following the best policy. To focus computation on the relevant state values, we introduce the idea of *masking* an ADD.

Given an ADD $D$ and a set of relevant states $U$, masking is performed by multiplying $D$ by $\chi_U$, which is the characteristic function of $U$. This has the effect of mapping all irrelevant states to the value zero. We let $D_U$ denote the resulting *masked ADD*. Mapping all irrelevant states to zero can simplify an ADD considerably. If the set of reachable states is small, the masked ADD often has dramatically fewer nodes. Since the complexity of computations using ADDs is a function of the size of the ADDs, this can dramatically improve the efficiency of computation using ADDs.

The first phase of LAO* uses symbolic reachability analysis to determine the set of relevant states. The second phase updates the state evaluation function for these states only, using the technique of masking to focus computation on the relevant states. The algorithm maintains two kinds of global data structure; characteristic functions for sets of states and value functions for sets of states. The first are used to mask the second.

We summarize symbolic LAO* in Table 3, and give a more detailed explanation of its alternating phases in the following.

**Solution expansion.** In the solution expansion step of the algorithm, we perform reachability analysis to find the set of states $F$ that are not in $G$ (i.e., have not been "expanded" yet), but are reachable from the set of start states, $S^0$, by following the partial policy $\pi_G$. These states are on the "fringe" of the states visited by the partial policy. We add them to $G$ and add them to the set of states $E \subseteq G$ that are visited by the current partial policy. This is analogous to "expanding" states on the frontier of a search graph in heuristic search. It expands the partial policy in the sense that the policy will be defined for a larger set of states in the dynamic-programming step. We perform this reachability analysis using the same relational product operator reviewed in Section 2 (but not one one that uses least-cost existential quantification).

We note that our symbolic implementation of LAO* does not maintain an explicit search graph. It is sufficient to keep track of the set of states that have been "expanded" so far, denoted $G$, the *partial value function*, denoted $f_G$, and a *partial policy*, denoted $\pi_G$. For any state in $G$, we can "query" the policy to determine its associated action, and compute its successor states. Thus, the graph structure is implicit in this representation.

Because a policy is associated with a set of transition functions, one for each action, we need to invoke the appropriate transition function for each action when computing successor states under a policy. For this, it is useful to represent the partial policy $\pi_G$ in a different way. We associate with each action $a$ the set of states for which the best action to take is $a$ under the current policy, and call this set of states $\chi_{S_\pi^a}$. Obviously we have $\chi_{S_\pi^a} \cap \chi_{S_\pi^{a'}} = \phi$ for $a \neq a'$, and

$\cup_a \chi_{S^a_\pi} = G$. Given this representation of the policy, line 4 in Table 3 computes the set of successor states following the current policy using the *image* operator.

**Dynamic programming.** The dynamic-programming step of LAO* is performed using a modified version of the SPUDD algorithm. The original SPUDD algorithm performs dynamic programming over the entire state space. We modify it to focus computation on reachable states, using the idea of masking. Masking lets us perform dynamic programming on a subset of the state space instead of the entire state space.

The pseudocode in Table 3 assumes that DP is performed on $E$, the states visited by the best (partial) policy, although a larger or smaller set of states can be updated by LAO* [11]. Because $\pi_G$ is a partial policy, there can be states in $E$ with successor states that are not in $G$, denoted $E'$. This is true until LAO* converges. In line 13, we compute these states in order to do appropriate masking. To perform dynamic programming on the states in $E$, we assign admissible values to the "fringe" states in $E'$, where these values comes from the current value function. Note that the value function is initialized to an admissible heuristic evaluation function at the beginning of the algorithm.

With all components properly masked, we can perform dynamic programming using the SPUDD algorithm. This is summarized in line 17. The full equation is

$$f^a(\mathbf{X}) = c^a_E(\mathbf{X}) + \sum_{E'} T^a_{E \cup E'}(\mathbf{X}, \mathbf{X}') \cdot f'_{E'}(\mathbf{X}').$$

The masked ADDs $c^a_E$ and $T^a_{E \cup E'}$ need to be computed only once for each call to *valueIteration*() since they don't change between iterations. Note that the product $T^a_{E \cup E'} \cdot f'_{E'}$ is effectively defined over $E \cup E'$. After the summation over $E'$, which is accomplished by existentially abstracting away all post-action variables, the resulting ADD is effectively defined over $E$ only. As a result, $f^a$ is effectively a masked ADD over $E$, and the maximum $M$ at line 18 is also a masked ADD over $E$.

The residual in line 20 is computed by finding the largest absolute value of the ADD $(f_E - f'_E)$. We use the masking subscript here to emphasis that the residual is computed only for states in the set $E$. Dynamic programming is the most expensive step of LAO*, and it is not necessary to run it until convergence each time this step is performed. Often a single iteration gives the best performance. We extract a policy in line 22 by comparing $M$ against the action value function $f^a$ (breaking ties arbitrarily): $\forall s \in E \;\; \pi(s) = a \;\; \text{if} \;\; M(s) = f^a(s)$.

**Convergence test.** At the beginning of LAO*, the value function $f$ is initialized to the admissible heuristic $h$. Each time the value iteration is performed, it starts with the current values of $f$. Hansen and Zilberstein (2001) show that these values increase monotonically in the course of the algorithm; are always admissible; and converge arbitrarily close to optimal. LAO* converges to an optimal or $\epsilon$-optimal policy when two conditions are met: (1) its current policy

**Table 4.** Performance comparison of LAO* and SPUDD.

| Example | | | Reachability Results | | | Size Results | | | | Timing Results | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | LAO* | | | LAO* | | SPUDD | | LAO* | | | SPUDD |
| | $\|S\|$ | $\|A\|$ | $\|E\|$ | $\|G\|$ | reach | nodes | leaves | nodes | leaves | expand | DP | total | total |
| r1 | $2^{15}$ | 20 | 134 | 413 | $2^{15}$ | 65 | 12 | 1288 | 406 | 0.24 | 17 | 17 | 539 |
| r2 | $2^{20}$ | 25 | 3014 | 3281 | $2^{20}$ | 181 | 19 | 15758 | 4056 | 0.46 | 54 | 544 | 12774 |
| r3 | $2^{20}$ | 30 | 10442 | 33322 | $2^{20}$ | 6240 | 2190 | 9902 | 4594 | 57.71 | 1679 | 1738 | 10891 |
| r4 | $2^{35}$ | 30 | 383 | 383 | $2^{35}$ | 77 | 4 | NA | NA | 0.05 | 7 | 7 | > 20hr |

does not have any unexpanded states, and (2) the error bound of the policy is less than some predetermined threshold. Like other heuristic search algorithms, LAO* can find an optimal solution without visiting the entire state space. The convergence proofs for the original LAO* algorithm carry over in a straightforward way to symbolic LAO*.

### 4.2 Experimental Results

Table 4 compares the performance of symbolic LAO* and SPUDD on four randomly-generated MDPs. Because the performance of LAO* depends on the starting state, our results for LAO* are averaged over 50 random starting states.

A simple admissible heuristic function was created by performing ten iterations of an approximate value iteration algorithm similar to APRICODD [8] on an initial admissible value function created by assuming the maximum reward is received each step. The first few iterations are very fast because the ADD representing the value function is compact, especially when approximation is used.

LAO* achieves its efficiency by focusing computation on a subset of the state space. The column labeled *reach* shows the average number of states that can be reached from the starting state, by following any policy. The column labelled $|G|$ is important because it shows the number of states "expanded" by LAO*. These are states for which a backup is performed at some point in the algorithm, and this number depends on the quality of the heuristic. The better the heuristic, the fewer states need to be expanded before finding an optimal policy. The gap between $|E|$ and *reach* reflects the potential for increased efficiency using heuristic search, instead of simple reachability analysis.

The columns labeled "nodes" and "leaves", under LAO* and SPUDD respectively, compare the size of the final value function returned by LAO* and SPUDD. The columns under "nodes" gives the number of nodes in the respective value function ADDs, and the columns under "leaves" give the number of leaves. Because LAO* focuses computation on a subset of the state space, it finds a much more compact solution (which translates into increased efficiency).

The last four columns compare the running times of LAO* and SPUDD. Timing results are measured in CPU seconds. The total running time of LAO* is broken down into two parts; the column "expand" shows the average time

for policy expansion and the column "DP" shows the average time for value iteration. These results show that value iteration consumes most of the running time. This is in keeping with a similar observation about the original LAO* algorithm for flat state spaces. The time for value iteration includes the time for masking. For this set of examples, masking takes between 0.5% and 2.1% of the running time of value iteration. The final two columns show that LAO* is much more efficient than SPUDD in solving these examples. This is to be expected since LAO* solves the problem for only part of the state space. Nevertheless, it demonstrates the power of using heuristic search to focus computation on the relevant part of the state space. The running time of LAO* is correlated with $|G|$, the number of states expanded during the search, which in turn is affected by the starting state, the reachability structure of the problem, and the accuracy of the heuristic function.

We refer to a longer paper for a more detailed description of the symbolic LAO* algorithm and more extensive experimental results [20].

## 5 Conclusion and Future Work

We have described symbolic generalizations of A* and LAO* heuristic search that use decision diagrams to exploit state absraction. In showing how to implement A* using ADDs, we introduced a new ADD operator called least-cost existential quantification. This operator makes it possible to simultaneously compute the successors of a set of states, update their g-values, and detect and remove duplicate paths in graph search. Our symbolic generalization of LAO* integrates a reachability analysis algorithm adapted from symbolic model checking with the SPUDD dynamic programming algorithm for factored MDPs. To integrate these two approaches, we introduced the concept of *masking* an ADD. Masking provides a way to focus computation on the relevant parts of the state space – the hallmark of heuristic search.

We have described work in progress. Our contribution is to show how to use symbolic model-checking techniques for planning problems that require cost minimization. More work needs to be done to improve the performance of these algorithms, before we can fully evaluate this approach. Among the questions we are continuing to investigate, we highlight the following.

- Current techniques require encoding the problem state using Boolean variables. Different encodings may lend themselves more easily to state abstraction. Thus, an important question is how to find the best encoding. Another question is whether recent extensions of decision diagrams that allow non-Boolean variables could be useful.
- The bottleneck of algorithms that use decision diagrams is the need to represent the full transition relation, especially when computing the successors of a set of states. To address this problem, the model-checking community has developed techniques for *partitioning* the transition relation. These allow significant improvement in efficiency, and adapting these techniques to our more general search algorithms is likely to improve their performance too.

Another possibility worth exploring is whether the transition relation can be represented procedurally, as is usually done for deterministic state-space search.
- The symbolic approach exploits the structure of a problem to create useful state abstractions. But not all problems have the kind of structure that can be exploited by these techniques. Can we characterize the kind of problem structure for which this approach works well?
- The approach to state abstraction adopted in this paper is to aggregate states with the same value. It may be possible to develop approximation algorithms that aggregate states that have a similar value, achieving greater state abstraction in exchange for a bounded decrease in solution quality [8].

Presenting our symbolic generalizations of A* and LAO* in the same paper allows us to compare and contrast their performance. This leads to an interesting observation. At present, our symbolic implementation of A* cannot solve problems of nearly the same size as single-state A* can solve. Even the fifteen puzzle presents a challenge. By contrast, our symbolic implementation of LAO* can efficiently solve factored MDPs that are as large or larger than any reported in the literature. Given this observation, we find it interesting that the problems solved by both algorithms in our experiments have roughly the same size state space! (In fact, a Boolean encoding of the state of the Fifteen Puzzle has 64 variables, almost twice as large as the largest factored MDP considered in the literature.) In our view, this underscores the fact that work on A* search is very mature and techniques have been developed that allow quite large problems to be solved. By contrast, work on decision-theoretic planning is relatively immature and test problems are still quite small.

We hope that by looking at these two classes of problems together, we will be able to use techniques that have proved successful for one in order to improve our ability to solve the other.

**Acknowledgements.** This research was supported in part by NSF grant IIS-9984952 and NASA grant NAG-2-1463. We thank the developers of the SPUDD planner [7] for making their code available.

## References

1. Cimatti, A., Roveri, M., Traverso, P.: Automatic OBDD-based generation of universal plans in non-deterministic domains. In: Proceedings of the Fifteenth National Conference on Artificial Intelligence. (1998) 875 – 881
2. Cimatti, M., Roveri, M.: Conformant planning via symbolic model checking. Journal of Artificial Intelligence Research **13** (2000) 305–338
3. Edelkamp, S., Reffel, F.: OBDDs in heuristic search. In: German Conference on Artificial Intelligence (KI). (1998) 81–92
4. Edelkamp, S., Reffel, F.: Deterministic state space planning with BDDs. In: Proceedings of the 5th European Conference on Planning (ECP-99). (1999) 381–2

5. Jensen, R., Veloso, M.: OBDD-based universal planning for synchronized agents in non-deterministic domains. Journal of Artificial Intelligence Research **13** (2000) 189–226
6. Jensen, R.: OBDD-based deterministic planning using the UMOP planning framework. In: Proceedings of the AIPS-00 Workshop on Model-Theoretic Approaches to Planning. (2000) 26–31
7. Hoey, J., St-Aubin, R., Hu, A., Boutilier, C.: SPUDD: Stochastic planning using decision diagrams. In: Proceedings of the Fifteenth Conference on Uncertainty in Articial Intelligence. (1999) 279–288
8. St-Aubin, R., Hoey, J., Boutilier, C.: APRICODD: Approximate policy construction using decision diagrams. In: Proceedings of NIPS-2000. (2000)
9. Bertsekas, D.: Dynamic Programming and Optimal Control. Athena Scientific, Belmont, MA (1995)
10. Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Science and Cybernetics (**SSC-4**) 100–107
11. Hansen, E., Zilberstein, S.: LAO*: A heuristic search algorithm that finds solutions with loops. Artificial Intelligence **129** (2001) 35–62
12. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers **C-35** (1986) 677–691
13. K.L. McMillan: Symbolic Model Checking. Kluwer Academic Publishers, Norwell Massachusetts (1993)
14. R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, F. Somenzi: Algebraic Decision Diagrams and Their Applications. In: IEEE /ACM International Conference on CAD, IEEE Computer Society Press (1993) 188–191
15. Somenzi, F.: Binary decision diagrams. In Broy, M., Steinbruggen, R., eds.: Calculational System Design. Volume 173 of NATO Science Series F: Computer and Systems Sciences. IOS Press (1999) 303–366
16. Edelkamp, S.: Directed symbolic exploration in AI-planning. In: AAAI Spring Symposium on Model-based Validation of Intelligence, Stanford University (2001) 84–92
17. Somenzi, F.: CUDD: CU decision diagram package. ftp://vlsi.colorado.edu/pub/ (1998)
18. Daniele, M., Traverso, P., Vardi, M.: Strong cyclic planning revisited. In: Proceedings of the 5th European Conference on Planning (ECP-99). (1999)
19. Nilsson, N.: Principles of Artificial Intelligence. Tioga Publishing Co., Palo Alto, CA (1980)
20. Feng, Z., Hansen, E.: Symbolic heuristic search for factored Markov decision processes. (2002) Proceedings of the 18th National Conference on Artificial Intelligence (AAAI-02).

# On the Construction of Human-Automation Interfaces by Formal Abstraction

Michael Heymann[1] and Asaf Degani[2]

[1] Department of Computer ScienceTechnion, Israel Institute of Technology
heymann@cs.technion.ac.il

[2] NASA Ames Research Center, California
adegani@mail.arc.nasa.gov

**Abstract**. In this paper we address the problem of designing systems for human-automation interaction that insure satisfaction of a wide range of performance requirements (such as guaranteeing the safety and liveness of mission critical operations). Our approach is based on formal procedures that focus on the information provided to the user. We propose a formal methodology for constructing interfaces and corresponding user- manuals that is based on performing a systematic abstraction of the behavioral model of the system. The procedure is aimed at achieving two objectives: First, the interface must be correct in that with the given interface the user will be able to perform the specified tasks correctly. Secondly, the interface must be succinct. The paper discusses the underlying concepts and the formal methods for this approach. Two examples are used to illustrate the methodology. The algorithm for constructing interfaces that is proposed in the paper can be automated, and a preliminary software system for its implementation has been developed.

## 1 Introduction

Human interaction with automation is so widespread that almost every aspect of our lives involves computer systems, information systems, machines, and devices. These machines are complex and are comprised of many states, events, parameters and protocols. User interfaces for such machines always present a (highly) reduced description of the underlying machine's behavior.

In the majority of today's automated systems, the human is the supervisor. Users interact with systems or tools to achieve specified operational tasks (Parsuramann et al., 2000) such as the execution of specific sequences of actions (e.g., a procedure for setting up a medical radiation machine), monitoring a machine's mode changes (e.g., an automatic landing of an aircraft), or preventing a machine from reaching specified illegal states (e.g., tripping a power grid). To achieve these task specifications, the user is provided with information about the behavior of the machine by means of an interface and associated user-manuals and other training material.

Naturally, for the user to be able to interact with the machine correctly and reliably so as to achieve the task specification, the information provided to the user about the machine must first and foremost be correct. Yet, while correct interaction can, in principle, always be achieved by providing the user with the full detail of the machine behavior, the amount of detail is generally unmanageable. Therefore, in practice, the

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 99-115, 2002.

interface and related user manuals are always a reduced, or abstracted, description of the machine's behavior, and a major concern of designers of automated systems is to make sure that these abstracted interfaces and manuals are adequate and correct.

Currently, the design decisions as to what information must be provided to the user, both in the interface and in user-manuals, are made intuitively. Systematic methodologies do not exist for these decisions and the resultant interfaces are sometimes either overly complex or flawed, leading to what is commonly called "automation surprises," where operators (e.g., pilots, technicians, users) have difficulty understanding the current status of an automatic system as well as the consequences of their interaction with it (Woods, Sarter, and Billings, 1997).

In an earlier paper (Degani and Heymann, 2002), we discussed a methodology for evaluating interfaces and user manuals. Given a description of the machine, specifications of the user's task, interface, and all relevant information the user has about the machine, the procedure evaluates whether the interface and user manual information are correct for the task. The proposed procedure can be automated and applied to the verification of large and complex human-machine systems.

In the present paper we take an additional step and discuss a formal methodology for automatic generation of correct and succinct interfaces and user manuals.

## 2 Formal Aspects of Human-Automation Interaction

We focus primarily on the information content provided to the user about the behavior of a system. This aspect of user interaction with machines can be described and analyzed formally by considering the following four elements: (1) the machine-model, (2) the operational tasks, (3) the machine's interface with the user, and (4) the user's model of the machine, i.e., the information provided to the user about the machine behavior (e.g., in the user manual).

### 2.1 Machine

The machines are modeled as finite state transition systems. A *state* represents a mode, or configuration, of the machine. Transitions represent discrete-state (mode) changes that occur in response to events that trigger them. Some of the transitions occur only if the user triggers them, while other transitions occur automatically and are triggered by the machine's *internal* dynamics, or its *external* environment.

To illustrate a typical machine model, let us consider the machine of Figure 1, which describes a simplified multi-mode three-speed transmission system proposed for a certain vehicle. We use the convention that user-triggered transitions are described by solid arrows, while automatic transitions are depicted by dashed arrows. The transitions are labeled by symbols to indicate the (triggering) circumstances under which the machine moves from state to state. The transmission has eight states, or modes. These modes are grouped into three super-modes that represent manually switchable gears (or speeds): low, medium and high. The states within each speed represent internal torque-level modes. Thus there are torque modes $L1, L2, L3$, in the low speed super mode; there are torque modes $M1, M2,$ in the medium speed

super mode; and modes $H1, H2, H3,$ in the high speed super mode. The transmission shifts automatically between torque modes (based on torque, throttle, and engine and road speeds). The automatic up-shifts (to higher torque modes) are denoted by the event symbol $\delta$ and the automatic down-shifts by the symbol $\gamma$. The (user operated) manual speed changes, achieved by pushing a lever up or down, are denoted in the Figure by the event symbols $\beta$ and $\rho$, respectively. Pushing the lever up shifts to a higher speed and pushing down shifts to a lower speed. The transmission is initialized in the low torque mode $L1$ of the low speed (as indicated in the Figure by the free incoming arrow).

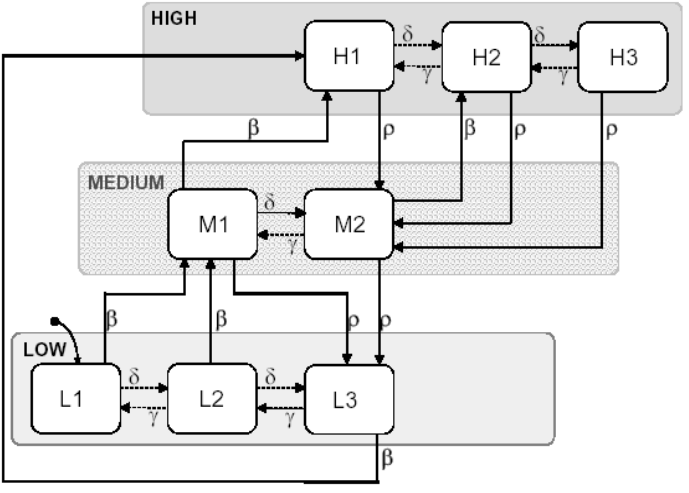



**Fig. 1.** Transmission system.

## 2.2 Task Specifications

The second element is the specification of the operational tasks the user is required to perform while using the machine. For example, a common task specification in an automated control system is that the user be able to determine unambiguously the current and the subsequent mode of the machine.

In terms of a formal description, the task specification to which we confine our attention in the present paper consists of a partition of the machine's state-set into disjoint clusters that we shall call specification classes (or modes) that the user is required to track unambiguously. In other words, does the user know whether the system is currently in, or is about to enter into, the super-mode High, Medium, or Low? We note that the user is not required to track every internal state change of the machine: for example, transitions between the modes $L1$, $L2$ and $L3$ inside mode Low.

### 2.3 Interface

The third element is the user interface. In practice, the interface consists of a control unit through which the user enters commands (e.g., mode selections, parameter changes) into the machine, as well as a display through which the machine presents information to the user. Generally, the interface provides the user a simplified view of the machine. Not all the events of the machine are annunciated to the user, and the interface displays only partial information about the actual behavior of the machine.

Formally, the interface consists of a listing and description of the events accessible to the user. These include, of course, all the user-triggered events (inputs to the machine), but generally only a subset of the events that are associated with automatic transitions. This is because some of the latter are not monitored at all, and others are monitored only in groups. The interface annunciation tells the user only that one of the events in the group took place, without specifying which.

To illustrate, let us return to the multi-mode transmission model of Figure 1. The system in Figure 2 gives one possible user interface for this model. Here the monitored events are only the ones triggered by the user. In the Figure 2 we have also provided a description of the three display modes, as well as how the user would observe the machine's behavior when all automatic transitions are internalized and unobserved. Note that the torque modes are completely suppressed from view.

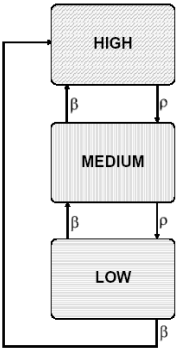



**Fig. 2.** Proposed interface and user model.

### 2.4 User Model

As mentioned earlier, the interface provides the user with a simplified view of the machine, in that it displays only partially the machines internal behavior. The description of the machine's operation that is provided to the user is generally also an abstracted simplification of the actual machine behavior. This description is usually provided in terms of a user manual, training material, formal instruction, or any other means of teaching the user; however, it is presented here as a formal model that we refer to as the *user model* of the machine. By its very nature, the user-model is based on the interface through which the user interacts with the machine, and thus relates to the modes and events that are displayed there. Therefore, for analysis purposes the

interface events and modes are all explicitly referred to in the user-model, and in this respect can be thought of as "embedded" in the user-model.

Let us examine the user interface displayed in Figure 2. This Figure depicts a possible user-model associated with the interface that monitors only the user-triggered events of the transmission system. This particular user-model has been obtained from the machine model of Figure 1 by suppressing (internalizing) the events that are not monitored, and grouping the states as suggested by the specification. It can be seen that the manual shifts from *MEDIUM* up to *HIGH* or down to *LOW,* as well as the down-shift from *HIGH* to *MEDIUM,* are always completely predictable. However, the up-shift from the LOW gear depends on the current torque mode. Note that the up-shifts from L1 and L2 switch the transmission to *MEDIUM* speed, while the up-shift from L3 switches the transmission to the *HIGH* speed. Therefore, from the suggested interface of Figure 2, it cannot be predicted whether the up-shift will lead the transmission from *LOW* to *MEDIUM,* or to *HIGH* gear. We must conclude that the user-model is inadequate for the task.

An alternate user-model for the transmission model that may remedy the above mentioned problem is presented in Figure 3. This user-model describes an interface that also monitors the occurrences of two specific automatic transitions, in addition to all user-actuated events. This user-model, in particular, is aimed at enabling the operator to determine whether the transmission is in a display-mode *LOW-1* (where an up-shift is supposed to lead to *MEDIUM* speed), or in the display-mode *LOW-2* (where an up-shift leads to *HIGH*).

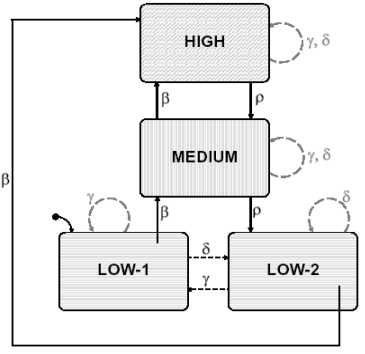



**Fig. 3.** Alternate interface and user model.

However, although the alternative user model of Figure 3 appears to have solved the problem, a formal verification employing the methodology recently proposed by Degani and Heymann (2000; 2002) shows that this user-model is also inadequate.

It is of course possible to try out other interfaces and user-models and then employ the verification procedure to determine their correctness. However, such an approach is not likely to be very fruitful: It may take considerable effort to develop and verify one design after the other, with no guarantee of success. Furthermore, even when a correct interface is found, there is no assurance that it is the simplest.

## 3 Machine Model Reduction

As mentioned earlier, one possible choice of user model is to take the full machine model as user model and the complete machine event set as the set of monitored events. If the machine model is deterministic (as we assume throughout this paper), this will insure that there will never be any problem in predicting the next state of the machine. But the operator would be required to track every state and every event in the machine – a formidable and impractical job. In the simple example of Figure 1, the machine has 8 states, 18 transitions and 4 distinct transition labels. But this is a tiny number when compared to "industrial size" situations.

In the present section we shall describe a procedure for the generation of *all* optimal user models and interfaces for a given machine model and task specification. In particular, we shall consider the problem of constructing, for a given machine and task specification, the set of all *best* possible user-models and event abstractions that satisfy the specification. Here, by *best* user models and interfaces we mean the ones that cannot be further reduced! Since, as we shall see, these user models (and associated event abstractions) are generally not unique, we cannot speak of user-model "synthesis," but rather, of machine model *reduction*. We shall show how all "smallest" user models and associated interfaces can be derived.

### 3.1 Compatible State Sets and Covers

We assume that the machine-model is given as a state machine and that the task specification is given as a partition of the state-set into disjoint classes of states that we refer to as *specification classes* (Heymann and Degani, 2002). Thus, each state of the machine model belongs to a unique specification class. (In Figure 1 which depicts the multi-mode three speed transmission, the specification classes consist of the three speeds; Low, Medium and High. Each state, or mode, belongs to exactly one speed.)

Let us consider a machine-model given as a state-machine, and let the task specification consist of a partition of the machine-model's state set $Q$ into disjoint specification classes $Q_1,...,Q_l$ (as described, for example, in Figure 1 where $l = 3$).

The user model must enable the user to operate the system correctly with respect to the specification classes. That is, it must enable the user to track the specification classes but not necessarily individual states. Thus, the user does not need to be able to distinguish (by means of the user model and interface) between two states $p$ and $q$ of the same specification class, if for the purpose of tracking the specification classes unambiguously it is sufficient for the user to know that the machine visited *either* $p$ or $q$. More explicitly, the user does not need to be able to distinguish between $p$ and $q$ if the specification class visited following any user-machine interaction starting in state $p$, is the same as the specification class visited following the same user-machine interaction starting at state $q$. This leads to the following definition: Two states, $p$ and $q$, are *specification equivalent* (or *compatible*), if given that the machine is presently in either state $p$ or $q$ (of the same specification class), the

specification classes to be visited under future inputs will be the same. Stated more formally, we have

**Definition:** Two states $p$ and $q$ are specification compatible if and only if the following two conditions both hold:

**1.** The states $p$ and $q$ belong to the same specification class,

**2.** If $p'$ and $q'$ are states such that there exists an even-string $s = \sigma_1 \ldots \sigma_n$ for which $p \xrightarrow{s} p'$ and $q \xrightarrow{s} q'$ are both defined, then $p'$ and $q'$ belong to the same specification class.

It is clear that if the only concern is to track the specification classes, two specification compatible states need not be distinguished in the user model. We may also conclude immediately that any set of states is specification compatible if all the pairs of states within that set are specification compatible.

Thus, if an efficient procedure is found for computation of all specification compatible pairs, the set of all compatible state sets will easily computed. Indeed, the compatible triples will be obtained as the state triples, all of whose pairs are compatible; compatible quadruples as the quadruples all of whose triples are compatible, and so on.

Next, we have the following:

**Definition:** A set C of compatible sets of states is called a cover of the state set of the machine-model, if every state of the machine-model is contained in one or more elements of C.

Since a set that consists of a single state is (trivially) compatible, it follows that every state is included in at least one compatible set, so that the set of all compatibles is always a cover.

**Definition:** A compatible set of states is called a *maximal* compatible set, if it is not a proper subset of another compatible set; that is, if it is not contained in a bigger compatible set of states.

Since sets that consist of a single state are compatible, it is clear that every state is contained in at least one maximal compatible set. It follows that the set of maximal compatibles is a cover.

**Definition:** A cover $C$ of compatibles is called a *minimal* cover, if no proper subset of $C$ is a cover.

Of particular interest to us will be the set of all minimal covers formed from the set of maximal compatibles. That is, we shall be interested in minimal covers whose component elements are maximal compatible sets. In general, the number of such minimal covers can be greater than one.

We shall see below that minimal covers by maximal compatibles constitute the foundation of the model reduction and interface generation procedure. However, we shall first show the set of compatibles is computed.

### 3.2 Generation of Compatible Pairs

As stated above, the computation of compatible sets hinges on the construction of the set of all compatible pairs. An efficient iterative algorithm for construction of compatible state pairs is based on the use of merger tables (see e.g., Paull and Ungar 1959, and Kohavi 1978, where related model reduction problems are discussed).

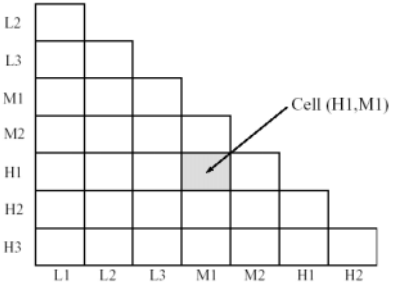



**Fig. 4.** Table of all pairs

A merger table is a table of cells representing distinct state pairs. An initial table for the eight states of our transmission example is shown in Figure 4. Each cell of the table corresponds to a pair of distinct states, and each pair of distinct states appears in the table exactly once.

Next, we have the following observations that can be easily derived from the definition of compatible pairs:

> A state pair $(p,q)$ of the same specification class is *compatible* if and only if for every event symbol $\sigma$ such that $p \xrightarrow{\sigma} p'$ and $q \xrightarrow{\sigma} q'$ are both defined, it is true that either $p'=q'$, or the pair $(p',q')$ is compatible.

We shall use the above characterization of compatible sets to obtain a complementary characterization of all pairs that are **not** compatible (or incompatible). It will then be convenient for us to compute recursively the set of all incompatible pairs. The set of compatible pairs will then consist of all state pairs that are not found to be incompatible. Based on the above characterization of compatible pairs, the characterization of incompatible pairs is as follows:

> A state pair $(p,q)$ is *incompatible* if and only if either $p$ and $q$ belong to distinct specification classes, or there exists an event symbol $\sigma$ for which

$p \xrightarrow{\sigma} p'$ and $q \xrightarrow{\sigma} q'$ are both defined, and the state pair $(p', q')$ is incompatible.

Using the above observations regarding compatible and incompatible pairs, the determination as to whether a state pair is compatible or incompatible is computed iteratively as follows.

1. For each state pair $(p, q)$ that can be determined as incompatible in the first step based on the above characterization (i.e., if $p$ and $q$ belong to distinct specification classes), we mark the corresponding cell *F* (for false). For all other state pairs, we write in their cells their associated *transition pairs* that consist of all distinct state pairs $(p', q')$ for which there exists an event symbol $\sigma$, such that the transitions $p \xrightarrow{\sigma} p'$ and $q \xrightarrow{\sigma} q'$ are both defined.

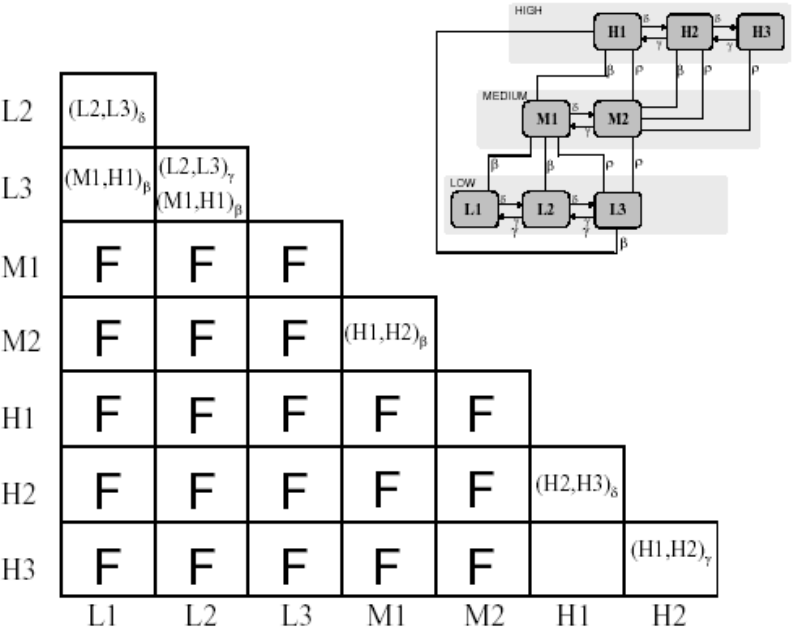



**Fig. 5.** Resolution table (initial).

For illustration, the initial resolution table for the transmission model of Figure 1 is presented in Figure 5. Notice that each transition pair in the table has been subscripted with the associated event label. This subscription is not essential to the algorithm and is for the reader's convenience only. Notice further that the cell *(H1,H3)* is empty because it is neither incompatible nor has associated transition pairs. Next, the table is resolved iteratively.

2. At each step of the iteration every state pair that has not yet been determined as *F* is updated as follows: If the cell of a state pair $(p,q)$ includes a transition pair $(p',q')$ whose cell has already been determined as *F* (incompatible), then the cell of $(p,q)$ is also denoted *F*. Otherwise, the cell of $(p,q)$ is modified as follows: Each transition pair $(p',q')$ in the cell of $(p,q)$ is replaced by all the transition pairs that appear in the cell of $(p',q')$.
3. If in a given iteration step no new incompatible state pairs are found (i.e., no new *F* designations are added to the table), then all the state pairs that are not designated as *F*, are given the designation *T* (for true). This completes the table resolution procedure and the determination of all compatible pairs.

To illustrate the iteration steps of the procedure, let us return to our transmission example. The table of Figure 6 is obtained from that of Figure 5 as follows: First we replace the transition pairs in the cell *(L1,L2)* by those in the cell *(L2,L3)*. The cells *(L1,L3)* and *(L2,L3)* are denoted with *F* because their cells include incompatible pairs. The remaining undecided state pairs (those that have not yet been given the value *F*) are modified according to the algorithmic procedure. For example, in the cell *(M1,M2)* we list the transition pairs from the table of Figure 5 of the cell *(H1,H2)* that consists of *(H2,H3)*.

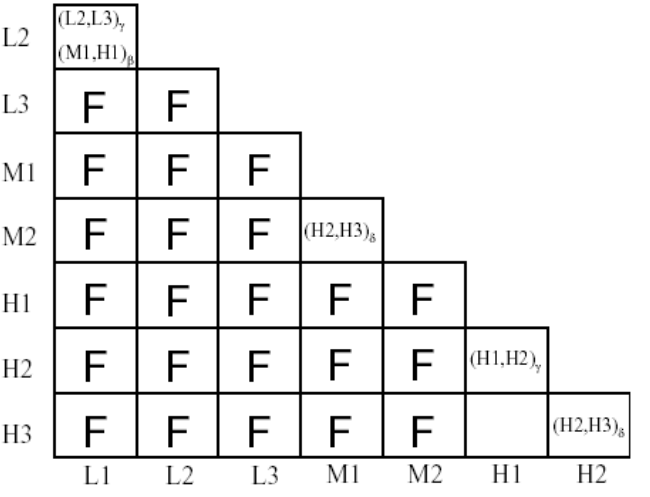



**Fig. 6.** Resolution table (after first iteration).

In the next resolution step the table of Figure 7 is obtained. Here the cell (L1,L2) is marked F upon substituting the value F of the cell (M1,H1,) which is incompatible. The remaining undecided cells are modified as specified by the algorithm. In fact, notice that no further change needs to be made to the table.

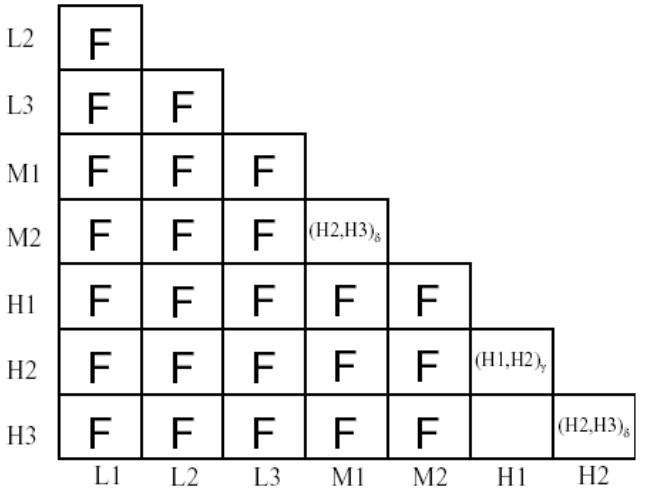



**Fig. 7.** Resolution table (after second iteration).

In the next step, no further incompatible pairs are created and the table remains identical to that of Figure 7. At this point, all the remaining undecided cells are marked T a shown in the table of Figure 8, concluding the table resolution.

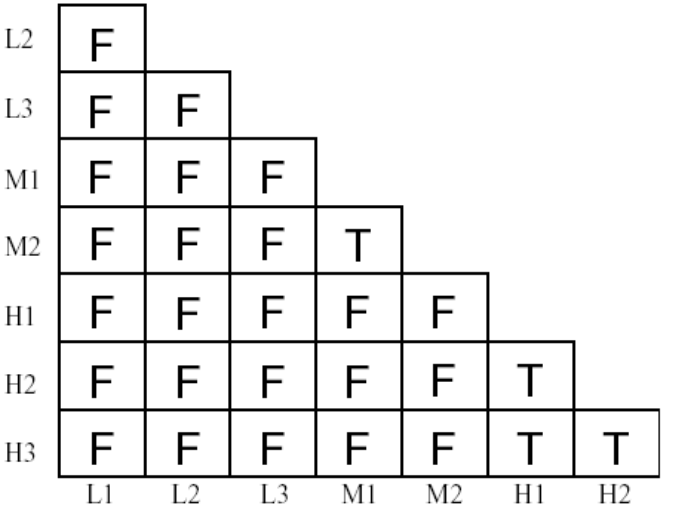



**Fig. 8.** Resolution table (completed).

Thus, as seen in Figure 8, for the example of Figure 1, the set of compatible pairs consists of *(M1,M2), (H1,H2), (H1,H3), and (H2,H3).* Notice that the states *L1, L2* and *L3* do not appear in any compatible pairs and therefore the singleton sets *(L1), (L2)* and *(L3)* are clearly maximal compatibles.

### 3.3 Generation of the Set of Maximal Compatibles

The procedure for generation of maximal compatibles consists of first systematically creating all compatible sets. We begin by computing all compatible triples, then compatible quadruples, then quintuples, and so on. A compatible triple is a triple all three of whose pairs are compatible; a compatible quadruple is a quadruple all of whose pairs are compatible, which is equivalent to a quadruple whose four triples are all compatible, and so on. Once all compatibles are listed, the maximal ones can easily be computed by deleting from the list all compatibles that are contained within larger ones.

For the transmission example, the maximal compatibles are easily found to be the sets *(L1), (L2), (L3), (M1,M2)* and *(H1,H2,H3)*. It is also not difficult to see that, in this case, they partition the state set into disjoint subsets and hence form the (unique) minimal cover by maximal compatibles.

### 3.4 Generation of Reduced Models

The generation of a reduced model that can serve as a correct user model for the given machine and specification is based on an abstraction of the machine-model. This reduced model is obtained by clustering the states into sets that consist of a minimal cover by maximal compatibles.

To this end, let us assume that a minimum cover consists of a given set of maximal compatibles $C_1,...,C_l$, where the set $C_i$, $i=1,...,l$, consists of states $\{q_{i_1},...,q_{i_{n_i}}\}$ of the machine model. The maximal compatibles $C_1,...,C_l$ form the state set of the reduced model. Here it is noteworthy that a minimal cover by maximal compatibles need not be a partition of the state set into disjoint subsets. Specifically, while each state of the machine model must be contained within some maximal compatible set, it may well be the case that a state is contained in more than one maximal compatible of the minimal cover. That is, these sets may (sometimes) have overlaps.

Next, we turn to computing the transitions in the reduced model. An event symbol $\sigma$ is said to be active at $C_i$, if there exists an outgoing transition in the machine model labeled by $\sigma$, at some state $q \in C_i$. That is, there exists a state $q'$ in the machine model, such that $q \xrightarrow{\sigma} q'$ is defined. We denote by $C_i(\sigma)$ the set of all states $q \in C_i$ for which an outgoing transition labeled by $\sigma$ exists.

Next, we define $S_i(\sigma)$ to be the set of all states $q'$ of the machine model, such that $q \xrightarrow{\sigma} q'$ for some $q \in C_i(\sigma)$. Thus, the set $S_i(\sigma)$ is the set of all states of the machine model that can be reached from states in $C_i$ through the event $\sigma$. It readily follows from the definition of compatible sets that there exists one or more element of $C_1,...,C_l$ which contain $S_i(\sigma)$. In the reduced model we then create a

transition labeled by $\sigma$ going from the state $C_i$ to the state $C_j$, where $C_j$ is the maximal compatible that contains $S_i(\sigma)$. If more than one such set $C_j$ exists, we can choose any one of these (and to avoid non-determinism in the reduced model we choose exactly one).

To summarize, the reduced model associated with the minimal cover $C_1,...,C_l$ is obtained as follows. The state set of the reduced model consists of elements $p_1,...,p_l$ (think of $p_i$ as associated with $C_i$). There is a transition labeled $\sigma$ from $p_i$ to $p_j$ if $C_j$ is the (chosen) set that contains $S_j(\sigma)$. The reduced model is initialized at state $p_k$ if the machine model is initialized at a state in $C_k$ (where, as before, there may be more than one possible selection if the initialization state is contained in more that one of the $C_i$). The reduced model obtained for the transmission example is shown in Figure 9.

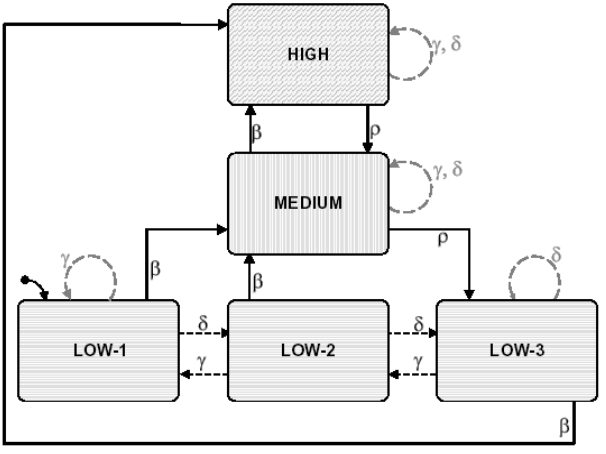



**Fig. 9.** The reduced user model.

### 3.5 Event Abstraction

The final step of the model reduction procedure consists of the abstraction of the reduced model's event set (when possible). Specifically, we ask which events can be internalized (i.e., need not be monitored) and which events can be clustered into groups so that instead of being monitored individually, they be monitored collectively. That is, the user will be informed that some events in the group occurred, but will not be informed which events of the group actually took place.

To this end the following abstraction rules apply:

1. An event can be internalized if it occurs in the reduced model only in self-loops.
2. A set of events can be grouped together, if every state transition that can be triggered by any event of the group can also be triggered by any other event of the group.

In the transmission example no event abstractions are possible. An illustration of event abstractions is provided in the example of the next section.

## 4 An Abstract Machine Example

In the above discussion on machine model reduction, we used an example of a transmission system. In this final section, we shall apply the reduction algorithm to a somewhat more complex machine. The machine in Figure 10 has nine states and 25 transitions. There are three specification classes: the gray region that includes states 7, 8, and 9; the wave-like region that harbors state 4 and 6; and the rest of the states of the machine (1, 2, 3, and 5). The task specification is similar to our previous one: the user has to track the machine along these three regions (or modes). Specifically, the user must be able to identify the current mode of the machine and anticipate the next mode of the machine as a consequence of his or her interactions.

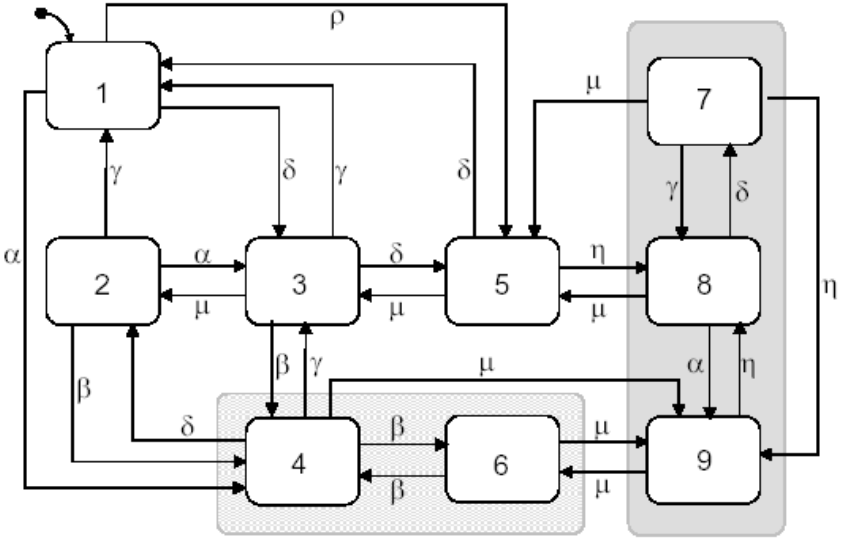



**Fig. 10.** An abstract machine model.

We perform the reduction procedure along the steps described in the previous section. First the table is constructed, and then the iterations are performed. The procedure terminates with only one minimal cover of maximal compatibles that consists of four state sets: (1,3,5) (2,3,5) (4,6) (7,8) and (9). Notice however, that this

example illustrates a case in which the cover is not a partition of the state set. Indeed, the state 3 is included in two distinct maximal compatibles.

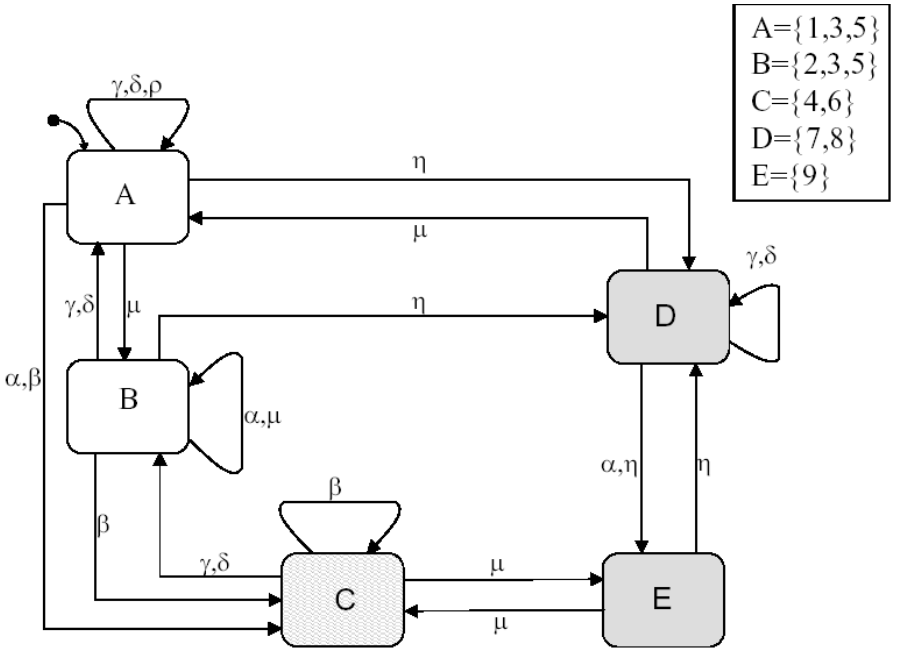



**Fig. 11**. Reduced model.

We then arbitrarily assign names to these sets, and call them A, B, C, D, and E, respectively. The reduced machine is obtained upon computation of the abstracted transitions as explained earlier (see Figure 11). It can be seen in this figure that the event $\rho$ occurs only in the self-loop in state A, and that the events $\gamma$ and $\delta$ are interchangeable. Thus, $\rho$ can be internalized and the events $\gamma$ and $\delta$ can be grouped. The result of this event abstraction is presented in the final reduced (user) model of Figure 12, which contains only 5 states and 16 transitions.

## 5 Conclusions

In this paper we discussed several formal aspects of the design of human-automation interaction. Specifically, we focused attention on the construction of user models and interfaces. Two objectives guided us in our design and analysis: (1) that the interfaces and user models be correct; and (2), that they be as simple as possible. We described a systematic procedure for generating such correct and succinct user-models and interfaces.

The proposed reduction procedure generates interfaces that are not necessarily intuitive or easily correlated with the underlying system (e.g., see the reduced user

model of Figure 12). Nevertheless, these user models are correct and efficient. They are also, irreducible.

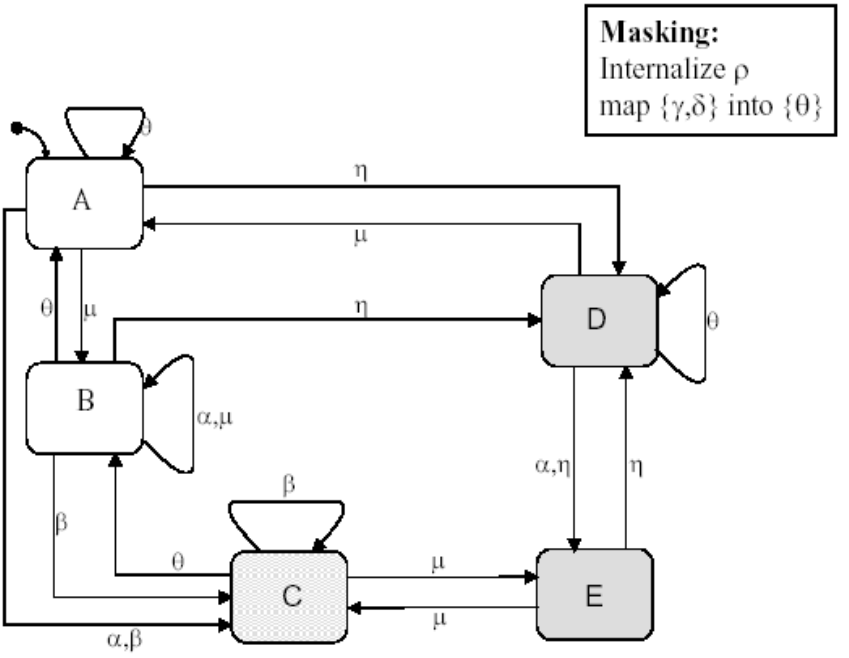



**Fig. 12.** Reduced model (with masking and internalization of events).

The proposed procedure may lead to more than one possible minimal (irreducible) interface and user-model. That is, it may find several minimal covers (of maximal compatibles). These minimal covers are all correct and efficient reductions of the same machine and task-specification. Naturally, the decision as to which one is selected constitutes a human-factors and/or engineering design decision. It affords the designer with several candidate interfaces and allows designers the freedom to choose the most appropriate one, given other design considerations such as Graphical User Interface considerations, users' preferences, and ease of implementation.

## References

Degani, A. and Heymann, M., Meyer, G., and Shafto, M. (2000). *Some Formal Aspects of Human-Automation Interaction, NASA Technical Memorandum 209600*, NASA Ames Research Center, Moffett Field, CA.

Degani, A. and Heymann, M. (2002). Formal Verification Of Human-Automation Interaction. *Human Factors*.

Heymann M., and Degani A. (2002). *On abstractions and simplifications in the design of human-automation interfaces.* NASA Technical Memorandum, NASA Ames Research Center, Moffett Field, CA.

Kohavi, Z. (1978). *Switching and Finite Automata Theory*. New York: McGraw-Hill.

Parasuraman, R., Sheridan, T.B., and Wickens, C.D. (2000). A model for the types and levels of human interaction with automation. *IEEE Transaction on Systems, Man, and Cybernetics – Part A: Systems and Humans, 30*(3), 286-297.

Paull, M.C. and Unger, S.H. (1959). Minimizing the number of states in incompletely specified sequential switching functions. *Institute of Radio Engineers Transactions on Electronic Computers*, 356-367.

Woods, D., Sarter, N., and Billings, C. (1997). Automation surprises. In G. Salvendy (Ed.), *Handbook of human factors and ergonomics* (pp. 1926-1943). New York: John Wiley.

# Pareto Optimization of Temporal Decisions

Lina Khatib*, Paul Morris, and Robert Morris

NASA Ames Research Center
MS 269-2
Moffett Field, CA 94035
{lina, pmorris, morris}@ptolemy.arc.nasa.gov

**Abstract.** The focus of this paper is on constrained optimization problems where the objective is to maximize a set of local preferences for times at which events occur. Previous work by the same authors has resulted in a reformulation of a subclass of these problems into a generalization of Temporal Constraint Satisfaction Problems, using a semi-ring as the underlying structure for reasoning about preferences. A tractable strategy for combining and comparing preferences was proposed, wherein computing global preferences consists of taking the minimum of the component preference values. This strategy, which we here dub the "weakest link optimization" (WLO) strategy, is a coarse method for comparing solutions, and therefore can easily be demonstrated to have drawbacks. To formalize this observation, we show here that the optimal solutions generated by WLO are not in general Pareto optimal. To compensate for these limitations, we make WLO more robust by combining it with a process that involves iteratively committing to temporal values in the set of optimal solutions, concomitantly fixing the preference value for the projection of the solution to the local preference. The intuition is simple: by applying WLO iteratively to reduced problems, WLO might be able to improve the preference values of the remaining temporal variables. We demonstrate the improvement gained from applying this technique on examples from Mars Rover planning domain, and prove that the combined strategy results in solutions that are in the set of Pareto optimal solutions.

## 1 Introduction

The notion of *softness* has been applied in the planning or scheduling literature to describe either a constraint or planning goal, indicating that either can be satisfied to matters of degree. For example, compare the goal *take the 7:30 train* with the soft goal *take the earliest train possible* (Krulwich, [4]). The soft goal can be viewed as imposing an ordering on a set of goals of the form *take the A train* based on the temporal value of $A$. Similarly, traditional scheduling problems can be viewed as inducing a preference ordering on the set of solutions, based, for example, on the makespan of the schedule.

* Kestrel Technologies

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 116–125, 2002.

The problem of planning with "soft goals" is the problem of finding strategies for making local decisions that lead to preferred solutions. Challenges arise when goals are *non-regressable*, meaning that the operators used to achieve the goals do not provide information that would assist the planner in meeting the desired criteria. For example, a blocks world soft goal "build the highest tower", is non-regressable with respect to standard strips operators for solving blocks world problems, since they do not include a parameter for measuring height. One strategy for overcoming this problem, (Krulwich, [4]) is to transform the soft goal into a hard one that meets the desired criteria (e.g., changing the goal to describe a specific $n$ block high tower for a $n$-blocks world problem), and then solving it. Another approach would be to devise heuristics that are guaranteed, or at least tend, to lead to preferred solution. For example, the *Longest Processing Time First* rule (LPT) in scheduling parallel machines is guaranteed to achieve a certain upper bound with respect to the ratio between the makespan achievable using it and the makespan of the optimal solution (Pinedo, [5]). Heuristics such as LPT order the decisions made by the scheduler, i.e., they order either which task to schedule next, or which resource (machine) to assign it to.

Sometimes softness is more naturally viewed as assigned to the constraints used in defining the problem domain. For example, in an earth orbiting spacecraft, sensitive instruments like imagers have *duty cycles*, which impose restrictions on the amount of use of the instrument. A duty cycle is typically a complex function based on both the expected lifetime of the instrument, as well as short term concerns such as the amount of heat it can be exposed to while turned on. Duty cycles impose constraints on the duration that the instrument can be on over a given length of time, but it is natural to view this duration as flexible. For example, this restriction might be waived to capture an important event such as an active volcano. Thus, the flexibility of the duty cycle "softens" the constraint that the instrument must be off for a certain duration. One way to express the soft constraint is to say that the duration of time the instrument is on should be as close as possible to that specified in the constraint. Reasoning about soft constraints for planning or scheduling is for the purpose of finding a solution that satisfies the constraint to the highest degree possible.

A dual to the problem of non-regressable soft goals is the problem of *non-compossible soft constraints*. This occurs when there is no principled means of ordering solutions based on the degree to which they mutually satisfy a set of soft constraints. For temporal reasoning problems, a simple method for evaluating the global temporal preference of a solution to a Temporal CSP based on local temporal preferences was introduced in (Khatib, *et. al.*, [3]) based on maximizing the minimally preferred local preference for a time value. Because the locally minimally preferred assignment can be viewed as a sort of "weakest link" with respect to the global solution , we dub this method "weakest link optimization" (WLO), in the spirit of the game show. WLO was chosen for reasons of computational efficiency. Specifically, its formalization as a semi-ring generalizing the TCSP framework leads to reformulation of Simple Temporal Problems (STPs), called STPs with Preferences (STPPs), that preserves the capability to

tractably solve for solutions. Unfortunately, as often occurs, this efficiency has a price. Specifically, WLO offers an insufficiently fine-grained method for comparing solutions, for it is based on a single value, viz., the "weakest link". It is consequently easy to conceive of examples where WLO would "miss" intuitively better solutions because of this myopic focus. Although it is possible to consider more robust alternatives to a WLO strategy for evaluating solutions, it is not clear whether any of these methods would preserve the computational benefits of WLO. This impasse is the starting point of the work described in this paper.

We propose an approach to making WLO more robust by combining it with a lookahead strategy for solving soft constraint reasoning problems. The process involves iteratively fixing temporal values for a STPP (at the same time fixing the preference value for the projection of the solution the local preference), and applying the WLO to the reduced problem that results from the commitment. The intuition is simple: by re-starting WLO iteratively on the reduced problem, WLO might be able to improve the preference values of the remaining temporal variables, thus compensating for the myopia of WLO. We demonstrate this technique on examples from Mars Rover planning domain.

The remainder of this paper is organized as follows. In Section 2, we summarize the soft constraint framework introduced previously; this review serves to motivate the current work. We then illustrate the deficiencies of WLO on a simple example, which also reveals the intuition underlying the proposed strategy for overcoming this deficiency. The main contribution of this paper is discussed in section 3, which formalizes this strategy and proves that any solutions generated by an application of this strategy is in the set of Pareto optimal solutions for the original problem.

## 2 Reasoning about Preferences with Soft Constraints

In (Khatib, *et. al.*, [3]), a *soft temporal constraint* is defined as a pair $\langle I, f\rangle$, where $I$ is a set of intervals $\{[a, b], a \leq b\}$ and $f$ is a function from $\bigcup I$ to a set $A$ of preference values. To compare and combine values from this set, $A$ is organized in the form of a *c-semiring* (Bistarelli, *et. al.*, [1]). A *semiring* is a tuple $\langle A, +, \times, 0, 1\rangle$ such that

- $A$ is a set and $0, 1 \in A$;
- $+$, the additive operation, is commutative, associative and 0 is its unit element;
- $\times$, the multiplicative operation, is associative, distributes over $+$, 1 is its unit element and 0 is its absorbing element.

A *c-semiring* is a semiring in which $+$ is idempotent (i.e., $a + a = a, a \in A$), 1 is its absorbing element, and $\times$ is commutative. Soft temporal constraints give rise to a class of constrained optimization problems called Temporal Constraint Satisfaction Problems with Preferences (TCSPPs). A TCSPP can simply be viewed as a generalization of a classical TCSP with soft constraints. In TCSPs (Dechter, [2]), a temporal constraint depicts restrictions either on the start times of events

(in which case the constraints are unary), or on the distance between pairs of distinct events (in which case they are binary). For example, a unary constraint over a variable $X$ representing an event, restricts the domain of $X$, representing its possible times of occurrence; then the interval constraint is shorthand for $(a_1 \leq X \leq b_1) \vee \ldots \vee (a_n \leq X \leq b_n)$. A binary constraint over $X$ and $Y$, restricts the values of the distance $Y - X$, in which case the constraint can be expressed as $(a_1 \leq Y - X \leq b_1) \vee \ldots \vee (a_n \leq Y - X \leq b_n)$. A uniform, binary representation of all the constraints results from introducing a variable $X_0$ for the *beginning of time*, and recasting unary constraints as binary constraints involving the distance $X - X_0$. A TCSP in which a single convex interval defines the constraint is called a Simple Temporal Problem (STP). As with classical TCSPs, the interval component of a soft temporal constraint depicts restrictions either on the start times of events, or on the distance between pairs of distinct events. The class of problems in which each constraint consists of a single interval is called *Simple Temporal Problems with Preferences* (STPPs). A *solution* to a TCSPP is a complete assignment to all the variables that satisfies the temporal constraints. An arbitrary assignment of values to variables has a *global preference value*, obtained by combining the local preference values using the semiring operations. A C-semiring induces a partial order relation $\leq_S$ over $A$ to compare preference values of arbitrary assignments; $a \leq_S b$ can be read *b is more preferred than a.* Classical Temporal CSPs can be seen as a special case of TCSPP, with "soft" constraints that assign the "best" (1) preference value to each element in the domain, and the "worst" (0) value to everything else. The optimal solutions of a TCSPP are those solutions which have the best preference value in terms of the ordering $\leq_S$. *Weakest Link Optimization* (WLO) is formalized via the semiring $S_{WLO} = \langle A, max, min, 0, 1 \rangle$. Thus, where $a, b \in A$, $a + b = max(a, b)$ and $a \times b = min(a, b)$, and 1 (0) is the best (worst) preference value.

Given a solution $t$ in a TCSPP with semiring $S_{WLO}$, let $T_{ij} = \langle I_{i,j}, f_{i,j} \rangle$ be a soft constraint over variables $X_i, X_j$ and $(v_i, v_j)$ be the projection of $t$ over the values assigned to variables $X_i$ and $X_j$ (abbreviated as $(v_i, v_j) = t_{\downarrow X_i, X_j}$). The corresponding preference value given by $f_{ij}$ is $f_{ij}(v_j - v_i)$, where $v_j - v_i \in I_{i,j}$. The global preference value of $t$, $val(t)$, is defined as $val(t) = min\{f_{ij}(v_j - v_i) \mid (v_i, v_j) = t_{\downarrow X_i, X_j}\}$.

As with classical (binary) CSPs, TCSPPs can be arranged to form a network of nodes representing variables, and edges labeled with constraint information. Given a network of soft constraints, under certain restrictions on the properties of the semiring, it can be shown that local consistency techniques can be applied in polynomial time to find an equivalent minimal network in which the constraints are as explicit as possible. The restrictions that suffice for this result apply to

1. the "shape" of the preference functions used in the soft constraints;
2. the multiplicative operator $\times$ (it should be idempotent); and
3. the ordering of the preference values ($\leq_S$ must be a total ordering).

The class of restricted preference functions that suffice to guarantee that local consistency can be meaningfully applied to soft constraint networks is called *semi-convex*. This class of functions includes linear, convex, and also some step

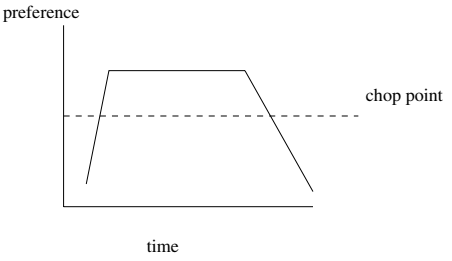



**Fig. 1.** "Chopping" a semi-convex function

functions. All of these functions have the property that if one draws a horizontal line anywhere in the Cartesian plane of the graph of the function, the set of $X$ such that $f(X)$ is not below the line forms an interval. Semi-convexity is preserved under the operations performed by local consistency (intersection and composition). STTPs with semiring $S_{WLO}$ can easily be seen to satisfy these restrictions.

The same restrictions that allow local consistency to be applied are sufficient to prove that STPPs can be solved tractably. Finding an optimal solution of the given STPP with semi-convex preference functions reduces to a two-step search process consisting of iteratively choosing a preference value, "chopping" each preference function at that point, then solving a STP defined by considering the interval of temporal values whose preference values lies above the chop line (semi-convexity ensures that there is a single interval above the chop point, hence that the problem is indeed an STP). Figure 1 illustrates the chopping process. It was shown that the "highest" chop point that results in a solvable STP is also the area at which the original STPP contains the optimal solutions. Binary search can be used to select candidate chop points, making the technique for solving the STTP tractable (see (Khatib, *et. al.*, [3]) for more details). The second step, solving the induced STP, can be performed by transforming the graph associated with this STP into a distance graph, then solving two single-source shortest path problems on the distance graph. If the problem has a solution, then for each event it is possible to arbitrarily pick a time within its time bounds, and find corresponding times for the other events such that the set of times for all the events satisfy the interval constraints. The complexity of this phase is $O(en)$ (using the Bellman-Ford algorithm).

## 3 The Problem with WLO

Formalized in this way, WLO offers a coarse method for comparing solutions, one based on the minimal preference value over all the projections of the solutions to local preference functions. Consequently, the advice given to a temporal solver by WLO may be insufficient to find solutions that are intuitively more

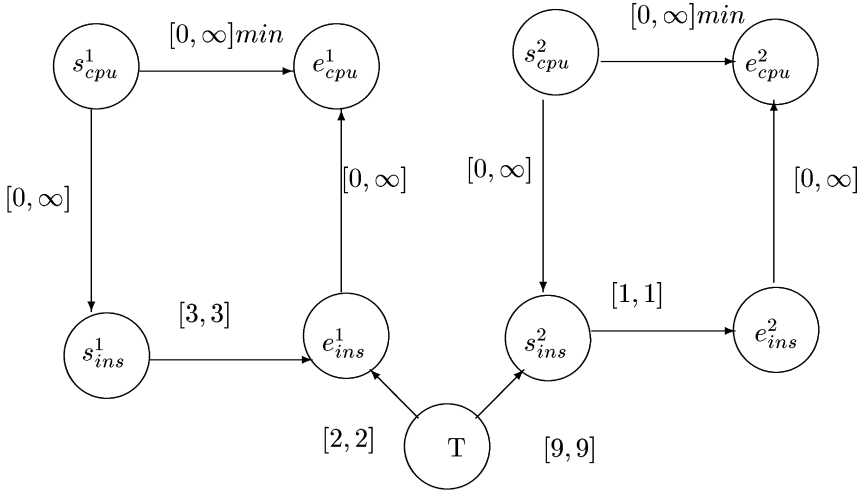



**Fig. 2.** The STPP for the Rover Science Planning Problem

globally preferable. For example, consider the following simple Mars rover planning problem, illustrated in Figure 2. The rover has a sensing instrument and a CPU. There are two sensing events, of durations 3 time units and 1 time unit (indicated in the figure by the pair of nodes labeled $s^1_{ins}, e^1_{ins}$ and $s^2_{ins}, e^2_{ins}$ respectively. There is a hard temporal constraint that the CPU be on while the instrument is on, as well as a soft constraint that the CPU should be on as little as possible, to conserve power. This constraint is expressed in the STPP as a function from temporal values indicating the duration that the CPU is on, to preference values. For simplicity, we assume that the preference function is the negated identity function, $pref(t) = -t$; thus higher values are preferred. Because the CPU must be on at least as long as the sensing events, any globally preferred solution using WLO has preference value -3. The set of solutions that have the optimal value includes both the solution in which the CPU is on for 1 time unit for the sensing event starting at $s^2_{ins}$, and the solution in which the CPU is on for 3 time units. The fact that WLO is unable to discriminate between the global values of these solutions despite the obvious fact that the first one is globally preferable to the other is a clear limitation of WLO.

One way of formalizing this drawback of WLO is to observe that a WLO policy is not *Pareto Optimal*. To see this, we reformulate the set of preference functions of a STPP, $f_1, \ldots, f_m$ as criteria requiring simultaneous optimization, and let $s = [t_1, \ldots, t_n]$ and $s' = [t'_1, \ldots t'_m]$ be two solutions to a given STPP. $s'$ dominates $s$ if for each $j$, $f_j(t_j) \leq f_j(t'_j)$ and for some $k$, $f_k(t_k) < f_k(t'_k)$. In a Pareto optimization problem, the *Pareto optimal set* of solutions is the set of non-dominated solutions. Similarly, let the *WLO-optimal set* be the set of optimal solutions that result from applying the chopping technique for solving STPPs described above. Clearly, applying WLO to an STPP does not guarantee

that the set of WLO-optimal solutions is a Pareto optimal set. In the rover planning problem, for example, the solution in which the CPU is on for 1 time unit dominates the solution in which it is on for 2 time units, but both are WLO-optimal, since they have the same weakest link value. [1] Assuming that Pareto-optimality is a desirable objective in optimization, a reasonable response to this deficiency is to replace WLO with an alternative strategy for evaluating solution tuples. A natural, and more robust alternative evaluates solutions by summing the preference values, and ordering them based on preferences towards smaller values (this strategy would be sufficient for ensuring Pareto optimality, since every minimal sum solution is Pareto optimal). This policy might be dubbed "utilitarian". The main drawback to this alternative is the inability to apply local consistency for improving search. The reason is that the formalization of utilitarianism as a semiring forces the multiplicative operator (in this case, *sum*), not to be idempotent (i.e., $a + a \not= a$), a condition required for ensuring that local consistency reduces the soft constraint reasoning problem to an equivalent one.

Of course, it would still be possible to apply a utilitarian framework for optimizing preferences, using either local search or a complete search strategy such as branch and bound. Rather than pursuing this direction of resolving the problems with WLO, we select another approach, based on an algorithm that interleaves flexible assignment with propagation using WLO.

## 4 An Algorithm for Pareto Optimization

The proposed solution is based on the intuition that if a constraint solver using WLO could iteratively "ignore" the weakest link values (i.e. the values that contributed to the global solution evaluation) then it could eventually recognize solutions that dominate others in the Pareto sense. For example, in the Rover Planning problem illustrated earlier, if the weakest link value of the global solution could be "ignored", the WLO solver could recognize that a solution with the CPU on for 1 time unit during the second instrument event is to be preferred to one where the CPU is on for 2 or 3 time units. We formalize this intuition by a procedure wherein the original STPP is transformed by iteratively selecting what we shall refer to as a *weakest link constraint*, changing the constraint in such a way that it can effectively be "ignored", and solving the transformed problem. A weakest link (soft) constraint is one in which the optimal value $v$ for the preference function associated with the constraint is the same as the optimal value for the global solution using WLO, and furthermore, $v$ is not the "best" preference value (i.e., $v < 1$, where 1 is the designated "best" value among the values in the semi-ring). Formalizing the process of "ignoring" weakest link values is a two-step process of committing the flexible solution to consist of the interval of optimal temporal values, and reinforcing this commitment by resetting their preferences to a single, "best" value. Formally, the process consists of:

[1] This phenomenon is often referred to in the literature as the "drowning effect".

- Squeezing the temporal domain to include all and only those values which are optimally preferred; and
- Replacing the preference function to one that assigns the highest (most preferred) value to each element in the new domain.

The first step ensures that only the best temporal values are part of any solution, and the second step allows WLO to be re-applied to eliminate Pareto-dominated solutions from the remaining solution space. The algorithm WLO+ (Figure 3) returns a Simple Temporal Problem (STP) whose solutions are precisely the WLO-optimal, Pareto-optimal solutions to the original STTP, $P$. Where $C$ is

Inputs: an STPP $P = (V, C)$
Output: An STP $(V, C_P)$ whose solutions are Pareto optimal solutions for $P$.
Begin
(1) $C_P = C$
(2) while there are weakest link soft constraints in $C_P$ do
(3) Solve $(V, C_P)$
(4) Select and delete every weakest link soft constraint from $C_P$
(5)For each deleted constraint $\langle [a, b], f \rangle$, add $\langle [a_{opt}, b_{opt}], f_{best} \rangle$ to $C_P$
(6) Return $(V, C_P)$
End

**Fig. 3.** STPP solver WLO+ returns a solution in the Pareto optimal set of solutions

a set of soft constraints, the STTP $(V, C_P)$ is solved (step 3) using the chopping approach described earlier. In step 5, we depict the soft constraint that results from the two-step process described above as $\langle [a_{opt}, b_{opt}], f_{best} \rangle$, where $[a_{opt}, b_{opt}]$ is the interval of temporal values that are optimally preferred, and $f_{best}$ is the preference function that returns the most preferred preference value for any input value. Notice that the run time of $WLO+$ is $O(|C|)$ times the time it takes to execute $Solve(V, C_P)$, which is a polynomial. To illustrate the main theoretical result of this paper, Figure 4 shows the relationship between the WLO-optimal solutions and the Pareto-optimal solutions. Of importance to the main result of the paper is that at least one WLO-optimal solution is Pareto-optimal. Otherwise, all WLO-optimal solutions will be dominated by Pareto-optimal ones; but this requires minimal values less than the weakest link value for some solution, contradicting the claim of WLO-optimality of the designated solutions. We now proceed to prove the main result, which is that the subset of solutions of the input STTP returned by WLO+ is a subset of the intersection of WLO-optimal and Pareto-optimal solutions. Formally, given an STTP P, let $Sol_{WLO}(P), Sol_{PAR}(P)$ be the set of WLO-optimal (respectively, Pareto-Optimal) solutions of P, and let $Sol_{WLO+}(P)$ be the set of solutions to $P$ returned by WLO+. Then

**Theorem 1.** *$Sol_{WLO+}(P) \subseteq Sol_{WLO}(P) \cap Sol_{PAR}(P)$. Moreover, if $P$ has any solution, then $Sol_{WLO+}(P)$ is nonempty.*

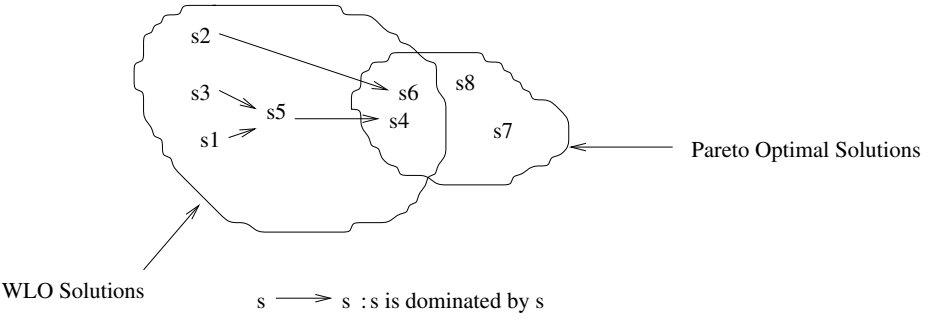



**Fig. 4.** Relationships between Solution Spaces for STTPs that are WLO or Pareto Optimal

*Proof.* First note that after a weakest link is processed in steps (4) to (6), it will never again be a candidate for a weakest link (since its preference is set to $f_{best}$). Thus, the algorithm will terminate when all the soft constraints in $C_P$ have preferences that have $f_{best}$ value over their entire domain. Now assume $s \in Sol_{WLO+}(P)$. Since the first iteration reduces the set of solutions of $(V, C_P)$ to $Sol_{WLO}(P)$, and each subsequent iteration either leaves the set unchanged or reduces it further, it follows that $s \in Sol_{WLO}(P)$. Now suppose $s \notin Sol_{PAR}(P)$. Then $s$ must be dominated by a Pareto optimal solution $s'$. Let $c$ be a soft constraint in $C$ for which $s'$ is superior to $s$. Thus, the initial preference value for $s$ on $c$ cannot be $f_{best}$. It follows that at some point during the course of the algorithm, $c$ will become a weakest link. Since $s$ survives until then and $s'$ dominates $s$, it follows that $s'$ also survives. At that time, $s$ will be excluded by step (6) since it is not WLO optimal, contradicting the assumption that $s \in Sol_{WLO+}(P)$. Hence, $s$ is in $Sol_{PAR}(P)$, and so in $Sol_{WLO}(P) \cap Sol_{PAR}(P)$. Next suppose the original STPP $P$ has at least one solution. To see that $Sol_{WLO+}(P)$ is nonempty, observe that the modifications in steps (4) to (6), while stripping out solutions that are not WLO optimal with respect to $(V, C_P)$, do retain all the WLO optimal solutions. Clearly, if there is any solution, there is a WLO optimal one. Thus, if the $(V, C_P)$ in any iteration has a solution, the $(V, C_P)$ in the next iteration will also have a solution. Since we are assuming the first $(V, C_P)$ $(= (V, C))$ has a solution, it follows by induction that $Sol_{WLO+}(P)$ is nonempty.□

**Corollary 1.** *If $P$ has any solution, the set $Sol_{WLO}(P) \cap Sol_{PAR}(P)$ is nonempty.*

This result shows that it is possible to maintain the tractability of WLO-based optimization while overcoming some of the restrictions it imposes. In particular, it is possible to improve the quality of the flexible solutions generated within an STPP framework from being WLO optimal to being Pareto optimal. The only other restriction still required is that of the semi-convexity of the preference functions. This restriction is needed because the "chopping" method assumes

that the domain above the chop point defines a STP, which implies that the preference function is semi-convex. A possible extension to this work would be to examine ways to relax this restriction in order to solve more general constrained optimization problems.

## 5 Summary

This paper has presented a reformulation of problems in the Pareto optimization of multi-attribute objective functions into a generalization of Temporal CSPs. The practical context from which this effort arose is temporal decision-making in planning, where associated with domains representing temporal distances between events is a function expressing preferences for some temporal values over others. The work here extends previous work by overcoming limitations in the approach that arose when considerations of efficiency in reasoning with preferences resulted in coarseness in the evaluation procedure for global temporal assignments.

**Acknowledgments.** The authors wish to thank Francesca Rossi and Jeremy Frank for helpful suggestions and comments.

## References

1. Bistarelli, S., Montanari, U., and Rossi, F. Semiring-based Constraint Solving and Optimization.*Journal of the ACM*, 44(2):201-236, 1997.
2. Dechter, R., Meiri, I. and Pearl, J. Temporal constraint networks. *Artificial Intelligence*, 49:61-95, 1991.
3. Khatib, L., Morris, P., Morris, R., and Rossi, F. Temporal Reasoning about Preferences. *Proceedings of IJCAI-01.*
4. Krulwich, Bruce. Planning for Soft Goals. *Proceedings of AIPS-92*, 289-290.
5. Pinedo, Michael. *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall, 1995.
6. Tsang, E. Foundations of constraint satisfaction. *Ed. Academic Press*, New York, 1993.

# An Information-Theoretic Characterization of Abstraction in Diagnosis and Hypothesis Selection

T.K. Satish Kumar

Knowledge Systems Laboratory
Stanford University
tksk@ksl.stanford.edu

**Abstract.** The task of model-based diagnosis is to find a suitable assignment to the behavior modes of components (and/or transition variables) in a system given some observations made on it. A complete diagnosis-candidate is an assignment of behavior modes to all the components in the system and a partial diagnosis-candidate is an assignment of behavior modes to only a subset of them. Corresponding to different characterizations of complete diagnosis-candidates (Bayesian model selection, consistency-based, model counting etc.), partial diagnosis-candidates play different roles. In the Bayesian model selection framework for example, they signify marginal probabilities, while in the consistency-based framework they are used to "represent" complete diagnosis-candidates. In this paper, we provide an information-theoretic characterization of diagnosis-candidates in a more general form — viz. "disjunction of partial assignments". This approach is motivated by attempting to bridge the gap between previous formalizations and to address the problems associated with them. We argue that the task of diagnosis actually consists of two separate problems, the second of which occurs more generally in hypothesis selection — (1) to characterize the space of complete or partial assignments (like in a posterior probability distribution), and (2) to abstract and approximate the information content of such a space into a representational form that can support tractable answering of diagnosis-queries and decision-making.

## 1 Introduction

Diagnosis is an important component of autonomy for any intelligent agent. Often, an intelligent agent plans a set of actions to achieve certain goals. Because some conditions may be unforeseen, it is important for it to be able to reconfigure its plan depending upon the state in which it is. This mode identification problem is essentially a problem of diagnosis. In its simplest form, the problem of diagnosis is to find the modes of behavior of components in a static system given some observations made on the system variables. Diagnosis in transition systems (trajectory selection) is in reality a simple extension of the above if we treat the transition variables as components (in one sense) [16]. Each trajectory is defined

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 126–139, 2002.

by an assignment to the transition variables, very much like the assignment of behavior modes to components in a static system.

The task of diagnosis arguably consists of two parts. The first part is to characterize the space of complete diagnosis-candidates by estimating the "goodness" of each of them. The second part is to abstract and approximate this space into a representational form that can support answering diagnosis-queries and decision-making tractably. A good example that illustrates these two phases is that of probabilistic reasoning and model selection using maximum likelihood. Given the description of a system in the form of a Bayesian network and some observations made on the system variables, we would like to make inferences about the behavior modes of components in the system. The first task is to calculate (at least conceptually) the posterior probability for any diagnosis-candidate (a candidate being an assignment of behavior modes to all the components of the system) conditioned on the observations made. The second task is to abstract this distribution into a form that can support tractable decision-making. In the case of model selection using maximum likelihood, this is simply to return the candidate with the highest posterior probability. Note that the abstraction phase is important because it is unreasonable to return the entire distribution (answering diagnosis-related queries or decision-making would then require us to either marginalize over, or find the maximum in, an exponential number of parameters).

In this paper, we argue that model selection using maximum likelihood (among other previous approaches) is justified only in limited cases — when there is a well-defined characterization of the space of variables over which we need to marginalize etc. A good illustration that drives home this argument is that of trajectory tracking. Here, we are given a physical system under continuous observations. The goal is to maintain a belief state and perform timely actions to achieve certain (re-)configurations. Typically, there exist many trajectories (possible paths of system states) that have reasonably high posterior probabilities. Choosing the best trajectory to guide our (re-)configuration actions need not always be the best thing to do. Actions that are instead directed by what is common among all good trajectories usually turn out to be much more fruitful. For example, if there are 10 trajectories all of which have probabilities around 0.1, then picking one of them (by possibly breaking ties arbitrarily) as a diagnosis is wrong with probability 0.9. Actions and (re-)configuration plans taken under this diagnosis can turn out to be misled and therefore extremely costly. Instead, taking actions based only upon the values of transition variables that occur commonly in all good trajectories may be much more useful [15].

In this paper, we formalize the above intuitive requirements and observations using an information-theoretic characterization of diagnosis-candidates. The first argument we make is that the primary goal of the abstraction phase in diagnosis is to return a representational form that can tractably support diagnosis-related queries and decision-making. We argue that a general form of diagnosis-candidates that can do this is a "disjunction of partial assignments". This representational form covers both complete diagnosis-candidates and par-

tial ones and gives us the framework for returning appropriate answers in situations where the space of complete or partial diagnosis-candidates do not contain them. Secondly, we introduce the notions of coverage and specificity for a diagnosis-candidate in the above form. The coverage of a candidate refers to the certainty with which it is true and the specificity refers to its information content. We provide an information-theoretic approach towards quantifying these two somewhat conflicting properties, while identifying the parameters that are specific to the application.

The organization of the paper is as follows. We first briefly present the background literature on the characterization of complete and partial diagnosis-candidates and some motivating examples that illustrate the shortcomings of the previous approaches. This leads us to the discussion of the notions of coverage and specificity of a diagnosis-candidate. We formalize these two parameters and provide an information-theoretic approach towards quantifying and comparing them. This allows us to unify the previous formalizations of complete and partial diagnosis-candidates and identify the application-specific parameters. We then briefly discuss some heuristic methods for finding good diagnosis-candidates according to our characterization and finally compare our work with related literature while alluding to some avenues for future work.

## 2 Background and Motivation

In this section, we review the previous approaches taken towards characterizing diagnoses. This would help us set the ground for the discussion of our work and help us provide a comparison of our approach with the previous approaches.

### 2.1 Characterizing Complete Diagnosis-Candidates

**Definition** *(Diagnosis System)***:** A *diagnosis system* is a triple ($SD$, $COMPS$, $OBS$) such that:
1. $SD$ is a system description expressed in one of several forms — constraint languages like propositional logic, probabilistic models like Bayesian network etc. $SD$ specifies both component behavior information ($SD_B$) and component structure information (i.e. the topology of the system) ($SD_T$).
2. $COMPS$ is a finite set of components of the system. A component $comp_i$ ($1 \leq i \leq |COMPS|$) can behave in one of several, but finite set of modes ($M_i$). If these modes are not specified explicitly, then we assume two modes — failed ($AB(comp_i)$) and normal ($\neg AB(comp_i)$).
3. $OBS$ is a set of observations expressed as variable values.

**Definition** *(Complete Diagnosis-Candidate)***:** Given a set of integers $i_1 \cdots i_{|COMPS|}$ (such that for $1 \leq j \leq |COMPS|$, $1 \leq i_j \leq |M_j|$), a *candidate* $Cand(i_1 \cdots i_{|COMPS|})$ is defined as $Cand(i_1 \cdots i_{|COMPS|}) = (\bigcup_{k=1}^{|COMPS|}(comp_k = M_k(i_k)))$.
Here, $M_u(v)$ denotes the $v^{th}$ element in the set $M_u$ (assumed to be indexed in some way).

**Notation:** When the indices are implicit or arbitrary, we will use the symbol $H$ to denote a candidate or a hypothesis.

The task of consistency-based diagnosis can be summarized as follows. Note that the definition of a diagnosis in this framework does not discriminate between single and multiple faults.
**Definition** *(Consistency-Based Diagnosis)***:** A candidate $H$ is a diagnosis if and only if $SD \cup OBS \cup H$ is satisfiable.

Consider diagnosing a system consisting of three bulbs $B_1, B_2$ and $B_3$ connected in parallel to the same voltage source $V$ under the observations $off(B_1)$, $off(B_2)$ and $on(B_3)$. $AB(V) \wedge AB(B_3)$ is a diagnosis under the consistency-based formalization of diagnosis if we had constraints only of the form $\neg AB(B_3) \wedge \neg AB(V) \rightarrow on(B_3)$. Intuitively however, it does not seem reasonable because $B_3$ cannot be *on* without $V$ working properly. One way to get around this is to include fault models in the system [29]. These are constraints that explicitly describe the behavior of a component when it is not in its nominal mode (most expected mode of behavior of a component). Such a constraint in this example would be $AB(B_3) \rightarrow off(B_3)$. Diagnosis can become indiscriminate without fault models. It is also easy to see that the consistency-based approach can exploit fault models (when they are specified) to produce more intuitive diagnoses (like only $B_1$ and $B_2$ being abnormal).

The technique of using fault models is associated with the problem of being too restrictive. We may not be able to model the case of some strange source of power making $B_3$ *on* etc. The way out of this is to allow for many modes of behavior for each component of the system. Every component has a set of modes (in which it can behave) with associated models. One of these is the nominal (or normal) mode and the others are fault modes. Each component has the *unknown* fault mode with the empty model. The unknown mode tries to capture the *modeling incompleteness assumption* (obscure modes that we cannot model in the system). Also, each mode has an associated probability that is the prior probability of the component being in that mode. Diagnosis can now be cast as a combinatorial optimization problem of assigning modes of behavior to each component such that it is not only consistent with $SD \cup OBS$, but also maximizes the product of the prior probabilities associated with those modes (assuming independence in the behavior modes of components).
**Definition** *(Combinatorial Optimization)***:** A candidate $H = \text{Cand}(i_1 \cdots i_{|COMPS|})$ is a diagnosis if and only if $SD \cup H \cup OBS$ is satisfiable and $P(H) = (\Pi_{k=1}^{|COMPS|} P(comp_k = M_k(i_k)))$ is maximized.

Another popular way of incorporating probabilities is when we have sufficient experience and statistical information associated with the behavior of a system. In such cases, the system description is usually available in the form of a probabilistic model like a Bayesian network instead of a set of constraints. Given some observations made on the system, the problem of diagnosis then becomes a Bayesian model selection problem.

**Definition** *(Model Selection using Maximum Likelihood)***:** A candidate $H$ is a diagnosis (for a probabilistic model of the system, $SD$) if and only if it maximizes the posterior probability $P(H/SD, OBS)$.

Yet another intuition behind characterizing diagnoses is the idea of explanation. Explanatory diagnoses essentially try to capture the notion of cause and effect in the physics of the system. The observations are asymmetrically divided into inputs ($I$) and outputs ($O$) [7]. The inputs ($I$) are those observation variables that can be controlled externally.

**Definition** *(Abductive Diagnosis)***:** An abductive diagnosis for ($SD$, $COMPS$, $OBS = I \cup O$) is a candidate $H$ such that $SD \cup I \cup H$ is satisfiable and $SD \cup I \cup H \rightarrow O$.

An approach that unifies all the previous approaches for characterizing complete diagnosis-candidates is that of model counting [13]. The model counting problem is the problem of estimating the number of solutions to a SAT (satisfiability problem) or a CSP (constraint satisfaction problem). It can be shown that model counting is a weaker form of probabilities and that probabilities provide only precision information over model counting [13]. It is natural then to use the expressions $\#(SD, E)/\#(SD)$ and $P(E)$ (for any event $E$) almost equivalently — except that we use the former when we do not know $P(E)$ explicitly. Here, $\#S$ indicates the number of solutions to a set of partially instantiated constraints $S$ (— instantiations of variable values correspond to observations). This framework can be used to unify and incorporate any kind of knowledge elements that we have without imposing restrictions on the representational form of the system description [13].

**Definition** *(Probability-Equivalent)***:** The *probability-equivalent* of $\#(SD, E)$ is defined to be $P(E)\ \#(SD)$ when $P(E)$ is given explicitly.

**Definition** *(Model Counting Characterization)***:** A diagnosis is a candidate $H$ that maximizes the number of consistent models to $(SD, OBS, H)$ using probability-equivalents wherever necessary.

### 2.2 Characterizing Partial Diagnosis-Candidates

So far, we have considered only the characterization of complete diagnosis-candidates. We now consider the characterization of partial diagnosis-candidates. Partial diagnosis-candidates are those that do not make a commitment to the behavior modes of all the components in the system, but only to a subset of them. The characterization of complete diagnosis-candidates forms a sufficient statistic for the characterization of partial diagnosis-candidates in the sense that given the information related to the space of complete diagnosis-candidates, partial diagnosis-candidates can be characterized without further referring to $OBS$ or $SD$. However, under different characterizations of complete diagnosis-candidates, partial diagnosis-candidates play different roles.

In the case of consistency-based and abduction-based approaches, partial diagnosis-candidates are used to "represent" complete diagnosis-candidates. The following set of definitions articulate these notions. For simplicity of discussion we will assume only two modes of behavior for all components — (normal ($\neg AB(c)$) and abnormal ($AB(c)$)).

**Definition:** An $AB-$*literal* is either $AB(c)$ or $\neg AB(c)$ for some component $c \in COMPS$.
**Definition:** An $AB-$*clause* is a disjunction of $AB-$literals containing no complementary pair of $AB-$literals.
**Definition:** A conjunction $C$ of $AB-$literals *covers* a conjunction $D$ of literals if and only if every literal of $C$ occurs in $D$.
**Definition:** A *partial diagnosis* for $(SD, COMPS, OBS)$ is a suitable conjunction $P$ of $AB-$literals such that for every satisfiable conjunction $F$ of $AB-$literals covered by $P$, $SD \cup OBS \cup \{F\}$ is satisfiable.

If $P$ is a partial diagnosis-candidate of $(SD, COMPS, OBS)$ and $C$ is the set of all components mentioned in $P$, then $\Pi_{c \in COMPS \setminus C} A(c)$ is a complete diagnosis-candidate, where each $A(c)$ is $AB(c)$ or $\neg AB(c)$. Thus a partial diagnosis $P$ represents the set of all complete diagnosis-candidates that contain $P$ as a sub-conjunct.
**Definition:** A *kernel diagnosis* is a partial diagnosis-candidate with the property that the only partial diagnosis-candidate that covers it is itself.
**Definition:** A *conflict* of $(SD, COMPS, OBS)$ is an $AB-$clause entailed by $SD \cup OBS$. A *minimal conflict* is a conflict no proper subset of which is is a conflict.

The partial diagnoses of $(SD, COMPS, OBS)$ are the implicants of the minimal conflicts of $SD \cup OBS$. The kernel diagnoses of $(SD, COMPS, OBS)$ are the prime implicants of the minimal conflicts of $SD \cup OBS$. In [7], some of the benefits of using kernel diagnoses are argued. However, there are several immediate disadvantages of such a characterization — (1) the number of kernel diagnoses may be exponential, (2) they make little sense in probabilistic domains, and (3) they disqualify diagnosis-candidates which in fact are the most intuitive to return (see next subsection for examples).

Under the Bayesian and model counting characterization of complete diagnosis-candidates, partial diagnosis-candidates signify marginalized probabilities over the missing variables. However, there are problems associated with even this perspective of partial diagnosis-candidates. Two partial diagnosis-candidates are comparable only if they assign values to the same subset of variables in the system, Given a diagnosis task, model selection is done by choosing the assignment that has the highest marginalized probability for a specified set of variables. However, there is no characterization of over what subset of variables we have to marginalize to be able to return the best candidates in that space. A significant drawback arising as an implication of this is in situations like trajectory tracking (as discussed before).

### 2.3 Motivating Examples: Notions of Coverage and Specificity

In this subsection, we present some scenarios where it is intuitively clear what the diagnosis should be, but none of the previous approaches towards characterizing complete or partial diagnosis-candidates capture them all.

Figure 1 shows 4 examples towards illustrating this. In each case, there are 4 components $A, B, C$ and $D$, and a plot showing the posterior probabilities

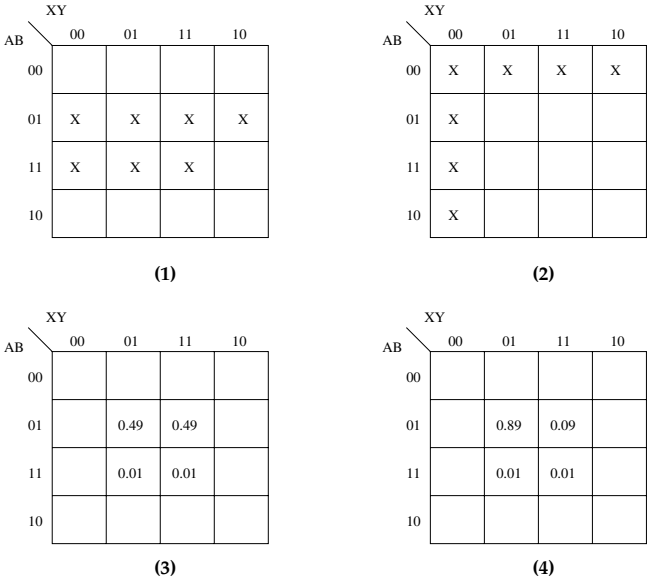



**Fig. 1.** Shows 4 examples to illustrate the abstraction phase in the task of diagnosis.

of the different combinations of their behavior modes (similar to a Karnaugh map). We use the notation $A$ to also mean $\neg AB(A)$ and $A'$ to mean $AB(A)$ (we assume only the normal or faulty mode for each component to keep the discussion simple). A real number in the Karnaugh map indicates the actual posterior probability of the corresponding complete assignment; a blank indicates a near-zero posterior probability and a $\times$ indicates uniform distribution of the posterior probability mass over all $\times$-ed cells. It is intuitively appealing that the following should be returned as the diagnoses in the 4 cases respectively: (1) $B$ (2) $A'B' \vee X'Y'$ (3) $A'BY$ (4) $A'BX'Y$. It is easy to note that the consistency-based characterization of diagnosis-candidates fails in cases (1), (3) and (4), and that model selection using maximum likelihood fails in (1), (2) and (3). The failures are due to the following reasons: (a) the consistency-based characterization disqualifies $B$ as being a valid candidate in (1), (b) the consistency-based characterization makes little sense in probabilistic domains like (3) and (4) — in particular, it would identify all models with non-zero probabilities, and (c) maximum likelihood does not automatically consider marginalization and must return a complete diagnosis-candidate.

The foregoing examples lead us to recognize the following two conflicting parameters of a diagnosis-candidate. The *specificity* of a diagnosis-candidate measures its information content. A complete diagnosis-candidate that makes a commitment to the modes of behavior for all the components is the most specific and provides maximum information. A partial diagnosis-candidate is less specific and contains lesser information since the behavior modes of certain components are not committed to. A disjunction of partial diagnosis-candidates is

even less specific than any of its constituent partial assignment clauses. The *coverage* of a diagnosis-candidate measures the certainty with which what we claim is true. The coverage of a candidate essentially corresponds to the fraction of consistent models of the system that also lie in the space of the models covered by the diagnosis-candidate. This factor is what is quantified in much of the present literature on trying to characterize diagnoses (including probabilities, model counting [13] etc.).

It is clear that when two candidates have the same specificity, the one with the higher coverage must be chosen and when they have the same coverage, the one with the higher specificity must be chosen. However, in most cases, such a comparison among the candidates is not straightforward because they might have different specificities and coverages. As an example, a complete diagnosis-candidate has the advantage that it has more information content, but a partial diagnosis-candidate although specifying lesser information, saves itself of the risk of easily going wrong. It is in such cases that we feel the necessity of contrasting the conflicting goals of coverage and specificity and weighing them appropriately using information-theoretic arguments.

Our first claim in trying to quantify coverage of a hypothesis against its specificity is that it is application specific. Consider a medical diagnosis scenario where the treatment of a patient is of much more concern than any costs associated with medical examinations. If there are 3 possible hypotheses $H_1$, $H_2$ and $H_3$ with probabilities 0.4, 0.4 and 0.2 respectively, then it is reasonable to report all of them (a disjunction of all of them) as the diagnosis. On the other hand, consider the scenario where a spaceship has the task of reconfiguring itself based on some belief of the possible worlds in which it is. It might be too costly if the wrong actions are taken. Reporting a disjunction of $H_1$, $H_2$ and $H_3$ is clearly not the choice of consideration. Instead, reporting what is common between $H_1$ and $H_2$ (like a certain engine is at fault, without committing to the state of a valve etc.) and taking actions based on this, may be much safer and more fruitful. The rest of the paper tries to provide a formalism for reasoning about different such scenarios. In particular, we show what parameters are application-specific and what the general theory behind the characterization of different diagnosis-candidates is. It should be noted however, that the examples in Figure 1 were chosen to be such that the coverage and specificity factors were lopsided, making them somewhat independent of the application.

## 3 An Information-Theoretic Characterization of Diagnosis-Candidates in DNF

The fundamental goal in the abstraction phase of diagnosis is to achieve computational tractability in answering diagnosis-queries and decision-making. Complete and partial diagnosis-candidates somewhat indirectly address this problem but are clearly unsuccessful (as illustrated in Figure 1). Assuming (without loss of generality) that we use the model counting characterization of complete diagnosis-candidates, the real question then becomes how well we can "simulate"

this space in a computationally efficient way. Note that we have two goals: (1) to come up with a representational form for a diagnosis-candidate that can make answering diagnosis queries tractable, and (2) to characterize what we mean by "simulating the space of complete diagnosis-candidates". This task relates to the notions of both specificity and coverage as discussed before. In the rest of the discussion, we will use model counting and probabilities somewhat interchangeably since the former can be shown to be a generalization of the latter [13].

As a solution to (1), we claim that a diagnosis candidate in a disjunctive form can simulate an approximate distribution over the space of complete assignments in a computationally efficient way. This is as follows: Given a DNF (disjunctive normal form), a complete assignment can be chosen by choosing a clause uniformly at random and then picking a satisfying assignment to it uniformly at random. Note however, that it is not computationally tractable to pick a satisfying assignment to the DNF uniformly at random. That is, a satisfying assignment that satisfies 2 clauses in the DNF has twice the probability of being picked up than one that satisfies just one clause. Also notice that we do not calculate the distribution of the space explicitly using this technique — it is just that it simulates such a distribution for answering diagnosis-queries. As an example, consider case (4) in Figure 1. If we choose $A'BY$ as a diagnosis, and have to estimate a complete assignment to all the modes, we would randomly pick between $A'BYX$ and $A'BYX'$ (extending on the clause $A'BY$). This marks a distribution of 0.5 and 0.5 for the two candidates respectively, and 0 for all others. We provide a treatment to (2) through the following definitions and characterizations.

**Definition:** A *clause* is an assignment of modes to a subset of components in the system. For example, $\neg AB(c_1)AB(c_2)$ is a clause.

**Definition:** A *diagnosis-candidate* is a disjunction of clauses. $\neg AB(c_1)AB(c_2) \vee AB(c_3)$ is a candidate.

Note that two candidates that are logically equivalent are not automatically taken to be the same.

**Definition:** The *coverage-space* of a candidate $C$ is defined to be the set of all consistent models of $C$ and is denoted by $CVG(C)$.

**Definition:** The *coverage* of a candidate $C$ (denoted by $cov(C)$) is defined as the weight of the probability distribution captured by any model in $CVG(C)$. That is, $cov(C) = \sum_{m \in CVG(C)} P(m)$.

Here, a model refers to an assignment of behavior modes to all the components.

**Notation:** We use $\dagger U$ to indicate the satisfiability of $U$. That is, $\dagger U = 1$ if $U$ is satisfiable and 0 otherwise. We use $\#S$ to indicate the cardinality of the set $S$.

**Definition:** The *induced distribution* of $C$ (with clauses $CL(C)$) over its coverage-space $CVG(C)$ is given by the probability distribution in which model $m_i$ of $CVG(C)$ has probability proportional to $\sum_{L \in CL(C)} (\dagger(m_i \cup L)/\#\{m_j | \dagger(m_j \cup L)\})$. This distribution is denoted by $D_{CVG(C)}$.

**Definition:** The *restricted distribution* of $C$ over its coverage-space $CVG(C)$ is denoted by $A_{CVG(C)}$ and indicates the actual probability distribution over $CVG(C)$.

**Definition:** The *information error* of a candidate $C$ (denoted $ierr(C)$) is defined as the $KL$ distance of $A_{CVG(C)}$ from $D_{CVG(C)}$.
This is a measure of both the specificity of $C$ and the amount by which we go wrong in the prediction of the distribution using $C$.
**Definition** *(Characterizing Diagnosis)***:** A candidate $C$ is a *diagnosis* when it minimizes $ierr(C) + W_{orc} \times (1 - cov(C))$.
Here $W_{orc}$ is some weighting factor that is application-specific.

### 3.1 Justification of Equations

The justification of the above quantification of specificity and coverage using application-specific parameters can be made using the following intuitive arguments employing oracles. An oracle is an entity that can be used alternatively to a diagnosis-candidate to estimate the posterior probability distribution of a space, but unlike the diagnosis-candidate, might have a representational form that makes it costly to use. The cost associated with using an oracle is therefore much more than the cost associated with paying for the information that we derive from it. On the other hand, a diagnosis-candidate by construction, has a representational form that can support easy estimation of the posterior probability distribution of the space it covers. Therefore, the amount of information by which we go wrong (measured by the $KL$ distance of $A_{CVG(C)}$ from $D_{CVG(C)}$) dominates the cost associated with using it. The cost associated with using the candidate is just the number of random bits we need (from our own internal system) to do sampling of consistent models. This is clearly negligible unless the number of clauses in the candidate is exponential. Now consider how much cost we incur in trying to estimate the actual distribution of posterior probabilities over the space of complete assignments given a diagnosis candidate $C$. We can estimate the distribution in $CVG(C)$ by bearing a cost proportional to $ierr(C)$. For the fraction of the space that is outside $CVG(C)$, that is $(1 - cov(C))$, we use the oracle and this bears a cost of $W_{orc}$. $W_{orc}$ is some application dependent parameter signifying how we should weigh the costs associated with using the oracle and the diagnosis-candidate.

Another way of thinking about the justification is by considering the $KL$ distance between the induced distribution of candidate $C$ and the actual distribution over the *entire* space of complete assignments. However, since the induced distribution is 0 for models that are not in the coverage-space of $C$, this distance is $\infty$ for all non-trivial candidates. In reality however, $\infty$ must reflect the maximum cost incurred for any wrong information in the application domain. If we use this parameter $W_{appl}$ instead of $\infty$, the modified $KL$ distance metric would be $\sum_m p(m) Min(log\frac{p(m)}{q(m)}, W_{appl})$. Assuming $W_{appl}$ is fairly large, it is now easy to note that this is equivalent to $ierr(C) + W_{appl}(1 - cov(C))$.

### 3.2 Behavior on Running Examples

In this subsection we show how the "intuitive" diagnoses for the 4 cases in Figure 1 are indeed captured by our characterization. In case (1), the candidate $B$ has

a very large coverage (maximum possible) and a very small *ierr* because of the uniformity on a majority of the models it covers. This accounts for it being able to minimize both the terms and hence is an appealing diagnosis-candidate. In case (2), again the coverage and *ierr* of $A'B' \vee X'Y'$ are very low. Note that by using a disjunction instead of just $A'B'$ or $X'Y'$ alone, the induced distribution will no more be exactly uniform over the coverage space because $A'B'X'Y'$ satisfies both the clauses $A'B'$ and $X'Y'$. However, this is very negligible compared to the gain in coverage. In case (3), $A'BY$ maximizes the coverage keeping the *ierr* low, and in case (4) the small gain in coverage by using $A'BY$ instead of $A'BX'Y$ is not justified because it significantly increases the *ierr* factor.

### 3.3 Bridging the Gap

We provide a brief analysis of the different characterizations of partial diagnosis-candidates based on the notions of coverage and specificity. Model selection using maximum likelihood over the space of complete diagnosis-candidates corresponds to choosing a model that has the "maximum coverage among all candidates that have the maximum specificity". When we marginalize probabilities over a space of variables $S$ and then employ maximum likelihood, this corresponds to choosing the candidate that has the "maximum coverage among all candidates that have the specificity specified by $S$". Prime and kernel diagnoses are notions that arise completely out of the necessity to represent the entire distribution over the space of complete assignments. It should be noted that unlike in maximum likelihood, our main concern here is to "maximize the coverage without worrying about the specificity".

### 3.4 Heuristic Algorithms

We provide a very brief allusion to two possible heuristic methods for recognizing good diagnosis-candidates under our formalization. One obvious algorithm would be to simply consider all subspaces (defined by any subset of components) and return the disjunction of assignments that have a "fairly large weight" in any of them. The second method is to consider only the space of complete diagnosis-candidates to choose the best few (based on some application-specific threshold) and try to represent them "compactly" using a Boolean formula.

## 4 Related Work

The literature related to the contents of this paper is along two fairly different lines. Along one of them is the work on various notions of diagnosis. This includes not only the characterization of complete and partial diagnosis-candidates, but also previous work on trying to unify model-based and probabilistic approaches in diagnosis. On the other line is the work in decision theory. Here, we are given a probabilistic model of the world and the stochastic effects of actions with associated utilities and costs. The goal is to be able to take actions that

maximize certain utility metrics. We provide a brief comparison of our work with both of these.

Characterization of complete and partial diagnosis-candidates in the consistency-based frameworks can be found in [7]. Fault models were used in [29] and abductive diagnoses are employed in [23], [20] etc. Combinatorial optimization is used as the method of choice in [16] and [30]. The combinatorial optimization formalization of diagnoses is attractive mainly because it allows for the use of computationally efficient methods [14]. Related work in trying to unify model-based and probabilistic approaches can be found in [24], [11], [17], [18] and [13]. [24] links abductive reasoning and Bayesian networks, and general diagnostic reasoning systems with assumption-based reasoning. [11] shows how to take results obtained by consistency-based reasoning systems into account when computing a posterior probability distribution conditioned on the observations (the independence assumptions are lifted in [18]). [17] gives a semantic analysis of different diagnosis systems using basic set theory. [13] gives a unification of the different notions of complete diagnosis-candidates in terms of model counting (i.e. estimating the number of satisfying assignments to a SAT or a CSP). Model counting algorithms can be found in [10] and [1].

In decision theory, we are given a set of actions with associated rewards in probabilistic domains. The goal is to perform actions that maximize the benefits (mainly in terms of expected gain in utility). The work presented in this paper differs from decision theory in the following important way. While decision theory deals with a characterization over the space of actions, our work is still concerned with a characterization over the space of models (assignments of behavior modes to components). It is important to reason over the space of models because it is often the case that for a given model as a diagnosis, we can come up with an elaborate plan to achieve a certain system configuration and this might potentially involve complex planning procedures. On the other hand, reasoning directly in the space of actions may involve not only having to deal with probabilistic information and other optimization metrics, but also cascading levels of causal interactions between actions to ensure that they make a valid plan. This makes it a much harder space to reason in. It is easier to first recognize the state of the world and decide what actions to take, rather than speak about all possible actions at the same time under uncertain conditions.

## 5 Conclusions

In this paper, we first argued that the task of diagnosis consists of two phases: (A) To characterize the space of complete assignments to behavior modes of components in a system. Here, we used the idea that the characterization of the space of complete diagnosis-candidates forms a sufficient statistic towards the characterization of any subspace — viz. the space of assignments to behavior modes of only a subset of the components. (B) To represent the information content in this space in a form that can make decision-making tractable. This problem occurs more generally in hypothesis selection. We introduced the idea

of using a "disjunction of partial assignments" as a diagnosis-candidate. This has several advantages. (1) It is a general form that can support tractable answering of diagnosis-queries and decision-making based on the argument that satisfying models can be sampled from a DNF in linear time. (2) It encompasses some "intuitive" diagnosis-candidates that are not otherwise properly captured using only the notions of complete or partial diagnosis-candidates. (3) It allows us to incorporate application-specific parameters based on the related notions of coverage and specificity. The coverage of a candidate refers to the certainty with which it is indeed the right model, and the specificity of a candidate refers to its information content. These two parameters were in general shown to be conflicting. (4) It allows us to bridge the gap between various formalizations of partial diagnosis-candidates (corresponding to different characterizations of complete diagnosis-candidates). Here, we claimed that the model counting formalization of complete diagnosis-candidates bridges the gap between various other characterizations of complete diagnosis-candidates [13]. (5) It forms an interesting comparison to decision-theoretic planning. We also discussed in the paper the "oracle" argument for information-theoretically quantifying the coverage versus the specificity of a diagnosis-candidate. Most of our future work is directed towards the development of computationally tractable algorithms to return "good" diagnosis-candidates based on our characterization of diagnosis, and a further investigation into the relationship of our work with decision theory.

## References

1. Bayard, R. J. and Pehoushek, J. D. 2000. Counting Models using Connected Components. Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000).
2. Console, L. and Torasso, P. 1991. A Spectrum of Logical Definitions of Model-Based Diagnosis. Computational Intelligence 7(3): 133-141.
3. Darwiche, A. 2001. On the Tractable Counting of Theory Models and its Applications to Belief Revision and Truth Maintenance. To appear in Journal of Applied Non-Classical Logics.
4. Dechter, R. 1992. Constraint Networks. Encyclopedia of Artificial Intelligence, second edition, Wiley and Sons, pp 276-285, 1992.
5. de Kleer, J. 1986. An Assumption Based TMS. Artificial Intelligence 28 (1986).
6. de Kleer, J. and Williams, B. C. 1987. Diagnosis with Behavioral Modes. Artificial Intelligence 32 (1987) 97-130.
7. de Kleer, J. Mackworth, A. K. and Reiter, R. 1992. Characterizing Diagnoses and Systems. Artificial Intelligence 56 (1992) 197-222.
8. Forbus, K. D. and de Kleer, J. 1992. Building Problem Solvers (MIT Press, Cambridge, MA, 1992).
9. Hamscher, W. Console, L. and de Kleer J. 1992. Readings in Model-Based Diagnosis. Morgan Kaufmann, 1992.
10. Karp, R. Luby, M. and Madras N. 1989. Monte-Carlo Approximation Algorithms for Enumeration Problems. Journal of Algorithms 10 429-448. 1989.
11. Kohlas, J., Anrig, B., Haenni, R., and Monney, P. A. Model-Based Diagnosis and Probabilistic Assumption-Based Reasoning. Artificial Intelligence, 104 (1998) 71-106.

12. Kumar, T. K. S. 2001. QCBFS: Leveraging Qualitative Knowledge in Simulation-Based Diagnosis. Proceedings of the Fifteenth International Workshop on Qualitative Reasoning.
13. Kumar, T. K. S. 2002a. A Model Counting Characterization of Diagnoses. Proceedings of the Thirteenth International Workshop on Principles of Diagnosis (DX 2002).
14. Kumar, T. K. S. 2002b. HCBFS: Combining Structure-Based and TMS-Based Approaches in Model-Based Diagnosis. Proceedings of the Thirteenth International Workshop on Principles of Diagnosis (DX 2002).
15. Kumar, T. K. S. and Dearden, R. 2002. The Oracular Constraints Method. Proceedings of the Fifth International Symposium on Abstraction, Reformulation and Approximation (SARA 2002).
16. Kurien, J. and Nayak, P. P. 2000. Back to the Future for Consistency-Based Trajectory Tracking. Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000).
17. Lucas, P. J. F. 1998. Analysis of Notions of Diagnosis. Artificial Intelligence, 105(1-2) (1998) 293-341.
18. Lucas, P. J. F. 2001. Bayesian Model-Based Diagnosis. International Journal of Approximate Reasoning, 27 (2001) 99-119.
19. MacKay, D. J. C. 1991. Bayesian Interpolation. Neural Computation 4(3): 415-447, 1991.
20. McIlraith, S. 1998. Explanatory Diagnosis: Conjecturing Actions to Explain Observations. Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98).
21. Mosterman, P. J. and Biswas, G. 1999. Diagnosis of Continuous Valued Systems in Transient Operating Regions. IEEE Transactions on Systems, Man, and Cybernetics, 1999. Vol. 29, no. 6, pp. 554-565, 1999.
22. Nayak, P. P. and Williams, B. C. 1997. Fast Context Switching in Real-time Propositional Reasoning. In Proceedings of AAAI-97.
23. Peng, Y. and Reggia, J. A. 1990. Abductive Inference Models for Diagnostic Problem Solving. New York: Springer-Verlag
24. Poole, D. 1994. Representing Diagnosis Knowledge. Annals of Mathematics and Artificial Intelligence 11 (1994) 33-50.
25. Raiman, O. 1989. Diagnosis as a Trial: The Alibi Principle, IBM scientific center (1989).
26. Reiter, R. 1987. A Theory of Diagnosis from First Principles. Artificial Intelligence 32 (1987) 57-95.
27. Shanahan, M. 1993. Explanation in the Situation Calculus. In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93), 160-165.
28. Struss, P. 1998. Extensions to ATMS-based Diagnosis, in: J. S. Gero (ed.), Artificial Intelligence in Engineering: Diagnosis and Learning, Southampton, 1988.
29. Struss, P. and Dressler, O. 1989. "Physical Negation" - Integrating Fault Models into the General Diagnosis Engine, in: Proceedings IJCAI-89 Detroit, MI (1989) 1318-1323.
30. Williams B. C. and Nayak, P. P. 1996. A Model-Based Approach to Reactive Self-Configuring Systems. In Proceedings of AAAI-96, 971-978.

# A Tractable Query Cache by Approximation

Daniel Miranker[1], Malcolm C. Taylor[2], and Anand Padmanaban[3]

[1] Department of Computer Sciences, University of Texas
Austin, Texas 78712
miranker@cs.utexas.edu
[2] Radiant Networks, Cambridge, UK
malcolm.taylor@radiantnetworks.co.uk
[3] Oracle Corp., Redwood City, CA
Anand.Padmanaban@oracle.com

**Abstract.** In this paper we present the organization of a predicate-based query cache suitable for integration with agent-based heterogeneous database systems. The cache is managed using a tractable (*sound and complete*) query containment algorithm, yet there are no language restrictions placed on the applications. This is accomplished by introducing query approximation.
Query approximation is a compilation technique where a query expression in a general query language is mapped to a query expression in a restricted language. We define a target language such that query containment can be tested in polynomial time. We specify the algorithms by which the query engine and the cache manager may negotiate a choice of approximation and the development of a query plan. We use two application workloads and the TPC-D benchmark queries to assess the value of the cache within the architecture of the InfoSleuth heterogeneous database system.

## 1 Introduction

The speed by which heterogeneous databases may gather data from component databases suffers from additive network and database latencies. One active approach to ameliorating this cost is to cache previous queries and/or to maintain materialized views (prefetching) of the component databases. A central element in these approaches is to compute query containment. That is given a pair of queries, Q1 and Q2, the first derived from an application, the second a representation of a data source, is the data comprising the answer set of Q1 contained in (subsumed by) the data in the answer set to Q2. If so then the answer set of Q2 may be used to compute the answer set of Q1. We would like to determine if Q2 contains Q1, written as Q2 $\supseteq$ Q1, by examining only the text of the query predicates and not rely on the materialization of the answer set or other physical properties (e.g. an address of a disk page).

The technical challenge for predicate caching as well as dealing with component DBs as views is that if the queries are expressed in any common query

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 140–151, 2002.

language, determining query containment is NP-hard. There have been many efforts to develop query languages where query containment is tractable [8,12,10, 3,16]. Although a number of useful results have been obtained, these languages place severe restrictions on the select and/or join conditions allowed in queries, and hence have limited applicability to the problem of predicate caching.

We define a query language that places few restrictions on the select/join conditions, while still allowing a tractable containment algorithm. We present a cache architecture which does not put any restriction on an application's queries, yet maintains a query cache based on tractable query containment. The basic idea is that there are precise syntactic structures that make query containment hard. In our system an application query is checked to see if it contains such a construct. If so, a new query is formulated by removing the problematic construct. The new query is a *weakened approximation* of the original. Thus, the new query is guaranteed to contain the original and evaluating query containment tractable. Only at the final stage of query processing are the omitted predicates applied, reducing the answer set to that of the original query.

This work was done as part of the Infosleuth mediator-based heterogeneous database system [9]. Since, in mapping queries to multiple data sources a mediator must resolve multiple sources of information and optimize cost, the integration of approximation fits naturally. If a query must be approximated there may be a number of ways to generalize the query, each with its own cost and intersection with cached queries. Further, if only some of the data needed to solve a query are cache resident, there can be a cost-based decision to determine whether the cached data should be merged with additional data or if the entire query should be reexecuted. In the spirit of agent architectures query plans are negotiated between the query agent and the cache agent.

Our hypothesis is that in real applications very few queries will require weakening and for those that do, the weakening does not appreciably add to the volume of data processed by the system. We support this hypothesis by examining two application workloads and TPC-D query benchmark and measuring the volume of additional data resulting from the weakening of those queries. Of the entire test suite, only 3 of 17 TPC-D queries are outside our tractable subset.

## 2 Tractability and Query Approximation

We exploit a narrow syntactic perspective of decidability wrt query languages. We start with queries in a general query language and identify individual predicates whose removal from the query, Q, yields a new query, Q" expressible in the restricted language. Using terminology developed in knowledge-compilation we call the new query an *approximation* of the original query [13]. Since the approximation was constructed by removing predicates, the approximation is guaranteed to contain the original query, Q' $\supseteq$ Q'. When this property holds the approximation is called an *upper bound*, designated $Q_{UB}$.

In our architecture the cached predicates are drawn only from a tractable query language. If an application submits a query that is more expressive than

the tractable language, the cache agent must determine an upper-bound approximation. A common problem formulation in knowledge-compilation is, given a in expression in an intractable logic, how hard is it to find a tractable least-upper bound. Determining a least-upper bound is often as hard as finding a direct solution. Since we are embedding this concept in a database system we may exploit the same cost-based infrastructure used to improve search in a query optimizer.

The query containment problem has been widely studied in the context of pure conjunctive queries. A pure conjunctive query corresponds to select-project-join queries of relational algebra, without additional built-in predicates. For pure conjunctive queries the containment problem can be reduced to one of finding a *containment mapping* from the variables of Q2 to those of Q1 [2]. A containment mapping is a mapping from the variables of Q2 to those of Q1, such that each literal in the body of Q2 is mapped to a literal in the body of Q1. The general containment problem has been shown to be NP-complete [2], but several researchers have identified restricted classes of pure conjunctive queries for which the problem can be solved in polynomial time. Saraiya [12] showed that query containment can be determined in linear time if no predicate appears more than twice in the sub-goals of Q1. Qian [10] developed a polynomial-time algorithm for the case where Q2 is acyclic. Subsequently, Qian's result was generalized by Chekuri and Rajaraman [3], to the case where the query width of Q2 is bounded by some integer k; the acyclic queries being those with query width 1.

In practical applications, queries frequently involve comparisons between a variable and a constant, as well as comparisons between two variables. The most common form of comparison is a test for equality, but inequalities ($<$, $\leq$, $>$, $\geq$) and disequations ($/=$) can also be used. Consequently, there have been investigations into whether the results concerning pure conjunctive queries can be extended to the case of conjunctive queries with built-in predicates. Klug [7] and van der Meyden [14] considered the effect of including inequalities, and showed that the problem becomes $\prod_2^p$-complete. Zhang and Ozsoyoglu [16] define a normal form for conjunctive queries with inequalities, which divides a query into two parts: a pure conjunctive query and a conjunction of equality and inequality comparisons Tests for containment can then be made by first looking for containment mappings between the pure conjunctive parts, and then checking containment of the comparison parts.

Kolaitis et al [8] considered the effect of allowing disequations as a built-in predicate. They show that the containment problem is in coNP if no database predicate appears in Q1 more than twice. They also show that the problem remains $\prod_2^p$-complete under the restriction that Q2 is acyclic.

Rosenkrantz and Hunt consider only conjunctions of equality and inequality comparisons (without the pure conjunctive component) and show that, as long as disequations are not allowed, *equivalence* of expressions can be deduced in polynomial time[11]. The family of comparisons considered in their work is more general than those considered by Zhang and Ozsoyoglu, in that it allows for a broad range of select and join conditions.

Chen and Roussopoulos [4] also omit the pure conjunctive component of a query. They do allow all equality and inequality comparisons, disjunctions as well as conjunctions. Their containment algorithm is sound but incomplete.

## 3 Algorithm for Query Containment

A polynomial query containment algorithm is presented for the simple restriction on conjunctive queries where no database predicate may appear more than once. This restriction does not impact most of the select and join conditions encountered in queries. Hence, most queries maybe approximated exactly. Our construction exploits a normal form similar to that used by Zhang and Ozsoyoglu, but using the more general form of comparisons considered by Rosenkrantz and Hunt. We adapt the approach of Rosenkrantz and Hunt to the problem of containment rather than equivalence. Thus, the containment problem can be solved in polynomial time even when allowing for a quite general class of comparisons involving both equalities and inequalities.

**Definition 1.** *(Tractable subset of SQL) A SQL query is in the tractable subset if it can be expressed as a conjunctive query with built-in predicates* $=, <, \leq, \geq, >$. *where each sub-goal of an approximation is expressed in terms of either a database predicate of a built-in predicate with the following restrictions:*

- *No database predicate may appear in more than one sub-goal.*
- *Each comparison involving the built-in predicates must take on of the following three forms:*
  1. `<variable> <comparison-op> <constant>`
  2. `<variable> <comparison-op> <variable>`
  3. `<variable> <comparison-op> <variable> + <constant>`

*Example 1.* The following query is in our tractable subset:
$q(X2, Y2) \leftarrow p(X1, X2, X3, X4) \wedge s(Y1, Y2, Y3) \wedge o(X1, Y1, Z1) =' Austin' \wedge Z1 \geq 10 \wedge X4 < X3 + 5$

The division into a pure conjunctive query and a conjunction of comparisons allows each part of the query to be dealt with separately; first seeking a containment mapping between the pure conjunctive components of the queries, and then examining each conjunction of comparison to establish containment. We show that each of these two parts can be done in polynomial time and we further prove that the algorithm is sound and complete.

**Definition 2.** *(Normal Form)A query Q is in* normal form *if it is expressed as* $Q_{PCQ} \wedge Q_{COMP}$, *where*

- $Q_{PCQ}$ *is a pure conjunctive query in which no variable appears more than once.*
- $Q_{COMP}$ *is a conjunction of comparisons that each take one of the forms "*$X \leq c$*" and "*$Z \leq Y + c$*", where* $X$ *and* $Y$ *are variables,* $c$ *is a constant, and* $Z$ *is either a variable or zero.*

**Algorithm 1** *(Containment of conjunctive queries)*

1. *convert each query to normal form*
2. *check whether there is a symbol mapping $\rho$ from Q2 to Q1*
3. *apply the mapping $\rho$ to the comparisons in Q2*
4. *check whether $Q1_{COMP}$ implies $\rho(Q2_{COMP})$*

**Lemma 1.** *Let Q be a query expressed in our tractable subset of SQL. Then Q can be converted to normal form in polynomial time.*

*Proof.* Suppose a variable X appears twice in $Q_{PCQ}$. We rename the second occurrence with a variable name that has not been used before (say, X1), and introduce an extra comparison "X = X1" in $Q_{COMP}$. Treating all repeated variables in this way will ensure that no variable appears more than once in $Q_{PCQ}$. Next, any comparison of form "X = Y" is replaced by a pair of comparisons "X $\leq Y \wedge Y \leq$ X". Then any comparison using $\geq$ (*or* $>$) can be rewritten in terms of $\leq$ (*or* $<$) by simply moving variables and/or constants from one side of the inequality to the other. Then, any comparison involving $<$ is rewritten in terms of $\leq$ by subtracting $\epsilon$ from the right-hand side, where $\epsilon$ is a value sufficiently small that no value between the new right-hand side and the old right-hand side can be represented on the computer. Finally, if a comparison has a constant on the left-hand side, the value of that constant is subtracted from each side so that zero becomes the left-hand side. □

*Example 2.* Let Q = E(X,Y), D(Y,Z,W), W>100. Then Q can be rewritten as E(X,Y), D(Y1,Z,W), Y≤Y1, Y1≤Y, 0≤W-100 - $\epsilon$

Having translated the queries to normal form, we first look for containment mappings between the pure conjunctive parts of the respective queries. The next two lemmas lead us to the result that our four-step algorithm is both sound and complete.

**Lemma 2.** *Let Q1 and Q2 be pure conjunctive queries, where Q1 has no repeated predicates. Then there cannot be more than one symbol mapping from Q2 to Q1.*

*Proof.* Consider any conjunct of Q2. There must be at most one predicate of the same name in Q1, since Q1 has no repeated predicates. So there is at most one way to map this conjunct of Q2. Similarly for all other conjuncts of Q2. Therefore, there is at most one symbol mapping from Q2 to Q1. □

**Lemma 3.** *(Zhang and Ozsoyoglu) Let Q1 and Q2 be two conjunctive queries in normal form. Let $\rho_1, \rho_2, ..., \rho_n$ be all the symbol mappings from Q2 to Q1. Then Q2 $\supseteq$ Q1 iff $n \geq 1$ and $Q1_{COMP} \Rightarrow \rho_1(Q2_{COMP}) \vee .... \vee \rho_n(Q2_{COMP})$*

**Theorem 1.** *Let Q1 and Q2 be two conjunctive queries in normal form. Let $M(Q2, Q1)$ denote the set of all symbol mappings from Q2 to Q1. If Q1 has no repeated database predicates, $Q2 \supseteq Q1 \Leftrightarrow \exists \rho \in M(Q2, Q1).Q1_{COMP} \Rightarrow \rho(Q2_{COMP})$.*

*Proof.* Follows from the Lemma 2 and Lemma 3. □

We now turn our attention to the conjunctions of comparisons. Rosenkrantz and Hunt [11] established that *equivalence* of conjunctions of comparisons can be determined in polynomial time. We adapt their approach to obtain a similar result for *containment*. Each query is represented as a weighted directed graph, in which there is one vertex for each variable and one vertex to represent zero. For each inequality of form "X ≤ c" there is an edge from the vertex of X to the vertex of zero, with weight c. For each inequality of form "Z ≤ Y + c" there is an edge from the vertex of Z to the vertex of Y, with weight c.

**Lemma 4.** *(Rosenkrantz and Hunt) If there is a path from the vertex of X to the vertex of Y, and the sum of the weights along the path is c, the comparisons in the predicate imply that* $X \leq Y + c$

**Lemma 5.** *(Rosenkrantz and Hunt) Let c be the weight of the shortest path from the vertex of X to the vertex of Y. Then the expression has a satisfying assignment in which* $X = Y + c$.

**Lemma 6.** *(Rosenkrantz and Hunt) An expression is satisfiable iff its graph has no negative weight cycles.*

**Theorem 2.** *Let P1 and P2 be conjunctions of comparisons from queries expressed in normal form. Then* $P2 \supseteq P1$ *iff for all variables X and Y, if the graph of P2 has a path from the vertex of X to the vertex of Y with weight c, then the graph of P1 has a path from the vertex of X to the vertex of Y with weight* $\leq c$.

*Proof.* proceeds by reductio ad absurdum.

Suppose the left-hand side is true but the right-hand side is false. Then there exist variables X and Y such that the shortest path from X to Y in P2 has weight d (say), while in P1 either there is no path from X to Y or else the shortest path has weight c, where d < c. If P1 has no such path, we can add an edge from X to Y with weight c (where c > d and c is greater than the negative of the shortest path (if any) from Y to X in P1). Then the new P1 is still satisfiable (since we have not created a negative weight cycle) and is still contained in P2. So now, without loss of generality, we assume that P1 has a path from X to Y of length c, where c > d. Therefore P1 has a satisfying assignment in which X = Y + c. But P2 implies X ≤ Y + d. And, since d < c, P2 is not satisfied by any assignment with X = Y + c. Therefore P1 is not contained in P2.

Suppose the left-hand side is false and the right-hand side is true. Then there exists an assignment that satisfies P1 but not P2. So we add this assignment to P2 and we should get a graph that is not satisfiable. (to add an assignment to a graph, we can take each variable assignment $X = x_1$, and add an edge from X to zero with weight $x_1$ and an edge from zero to X with weight $-x_1$). Since the new graph is not satisfiable, it must have a negative weight cycle. Let that cycle be $v_1, v_2, ..., v_n, v_1$. Now, by hypothesis, for each pair of adjacent vertices $v_i$, $v_{i+1}$ in this cycle, there is in P1 a path from $v_i$ to $v_{i+1}$ of lesser weight. By combining these lesser-weight paths between each pair of vertices in the cycle, we obtain a cycle in P1. Moreover, the total weight of this cycle in P1 must be less than or equal to the weight of the cycle in P2. Therefore P1 has a negative weight cycle, so P1 is not satisfiable. □

**Theorem 3.** *Let Q1 and Q2 be two conjunctive queries with built-in equality and inequality predicates such that Q1 has no repeated database predicates and all comparisons, in both Q1 and Q2, have one of the following forms:*

- `<variable> <comparison-op> <constant>`
- `<variable> <comparison-op> <variable>`
- `<variable> <comparison-op> <variable> + <constant>`

*Then the containment of Q1 in Q2 can be checked in polynomial time.*

*Proof.* We use Algorithm 1 to perform the check (by Theorem 1, this algorithm is sound and complete). The algorithm has four steps:

- By Lemma 1, the first step can be performed in polynomial time.
- The second step involves checking, for each predicate symbol in $Q2_{PCQ}$, whether that predicate symbol also appears in $Q1_{PCQ}$. It is clear that this can be done in polynomial time.
- The third step involves applying a symbol mapping to the variables of Q2. This again is clearly a polynomial-time operation.
- The fourth step involves computing the weights of the shortest paths between each pair of vertices. This can be done in cubic time[5].

Since each step is polynomial, the complete algorithm is polynomial. □

## 4 Agent-Based Negotiation of Query Plans

We address the architectural separation among the logic-related aspect of query approximation and containment in a manner consistent with cost-based decision making. The architecture encapsulates the logic and implementation of caching and the approximation method in the cache agent. The query planning components of the mediator are strictly responsible for choosing the caching plan. In the spirit of agent-based systems, we specify a distributed algorithm where the two agents negotiate a query plan.

In InfoSleuth application queries are executed as follows. Each resource agent (component database) advertises its existence to a mediator, specifying the capabilities of the agent and the ontological fragments for which it can provide information. User queries, expressed in terms of an ontology, are forwarded to the query agent. The query agent contacts the mediator and exploits the ontology and semantic analysis of query language expressions to obtain a list of the available resource that are capable of answering each subquery. The query agent assembles an evaluation plan, mapping subqueries to individual resources and generating a plan to fuse the results into the complete response. Typically resource agents are distributed over a wide-area network, making data transmission the dominant component of response time. If queries can be answered from cached data, remote access can be avoided and there is a concomitant reduction in response time.

The negotiation exploits the common breakdown of query optimization among logical planning and physical planning. Thus, negotiation is a two step

process. (See Figure 4.) The query agent submits a logical plan to the cache agent. The cache may be able to serve the data for the query but may not be able to answer the precise query. The cache agent responds to the query agent whether it can satisfy the query, in whole or in part. In the case that the cache can not provide a precise answer, the cache agent also informs the query agent that it will have to execute some additional predicates when fusing the data. The cache agent may further provide the query agent with cost metrics with which it may determine the quality of an individual plan and search for alternatives.

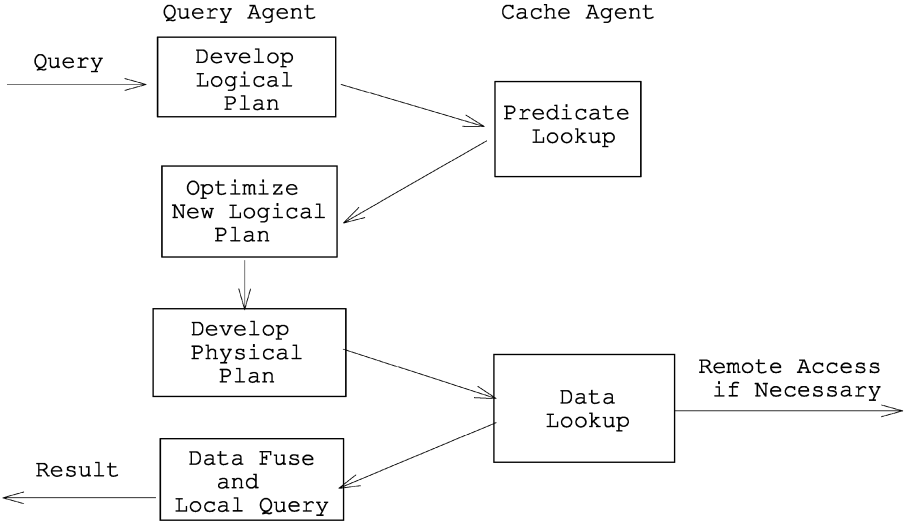



**Fig. 1.** Negotiation between Cache Agent and Query Agent

**Algorithm 2** *(Negotiation Algorithm)*
*The query agent asks, can you service query, $Q_i$ ? Let $C$ represent the content of the cache which is the union of all cached query predicates, $C = \bigcup_j Q_j^{UB}$.*

1. *Compute $Q_i^{UB}$. It is expected that most of the time $Q_i = Q_i^{UB}$.*
2. *Determine if and how $Q_i$ can be satisfied by the cache.*
3. *Return, 4 cases (see Table 1)*

In the simplest case, a request may be satisfiable, completely, from cached data. Thus, given a logical plan, the information returned to the query agent will be limited to the physical cost of satisfying the logical plan from cache or from a remote access. This comprises cases a) and b). In a more complex situation, the cache agent must also make a determination if only some of the data for the logical request can be answered locally. In that case the logical plan must be decomposed into a pair of new logical plans, one comprising a cache access, the

**Table 1.** Tests for query containment in the cache

| | Formal Property | Comment | Reply |
|---|---|---|---|
| a) | $Q_i \cap C = \phi$ | No data in the cache that will fulfill the query | Request will be fulfilled by remote request. |
| b) | $\exists Q_j^{UB} \in C$, $Q_i = Q_i^{UB} = Q_j^{UB}$ | There are data in the cache that precisely fulfills the query. | Request will be fulfilled, precisely, by the cache. |
| c) | $\exists Q_j^{UB} \in C$, $Q_j^{UB} \supseteq Q_i^{UB} \supseteq Q_i$ | The query is subsumed by a cached query | Request can be fulfilled by the cache, but query agent must execute query predicates, Q', where $Q' \supseteq Q_i - Q_j^{UB}$ |
| d) | $\exists Q_j^{UB} \in C$, $Q_i^{UB} \cap Q_j^{UB} \not= \phi$ $\wedge \neg(Q_j^{UB} \supseteq Q_i^{UB})$ | Only some of the data needed to fulfill the query are cached. | Request can be partially fulfilled by the cache, by $Q_j^{UB}$, the remainder fulfilled by querying the resource agent for, Q" $= Q_i^{UB} - Q_j^{UB}$, the query agent must execute query predicates, Q', where $Q' \supseteq Q_i - (Q_i \cap (Q_i^{UB} \cup Q_j^{UB}))$ |

other, a remote access. Since query decomposition is already implemented by the query agent, rather than duplicate this function in the cache agent, the cache agent will suggest to the query agent that it decompose the query accordingly. In essence, unless the query is satisfied by the cache precisely, the query agent and the cache agent negotiate how a query should be executed. The final decision is made by the query agent on a cost-basis. If costs dictate decomposition, the results are fused in precisely the fashion used to fuse any other pair of data sources. These elements comprise cases c) and d).

One obvious case where cost metrics appear likely to by-pass the cache is the execution of a relational select predicate on a table, when its evaluation using the cache comprises a long sequential scan, but its evaluation by the resource agent comprises a clustered index which promptly finds the result.

## 5 Empirical Results

We identified two query constructs that must be removed from the approximated query. One of these is the disequation predicate ($/=$) and the other is the repeated occurrence of a database predicate. Any comparison involving disequation is simply dropped from the query and applied later, after the data have been fetched from the component databases. Our claim is that this will have little affect on performance, because, disequations occur infrequently in practice and the selectivity of a disequation is typically much greater than that of an equality, or even an inequality, so dropping a disequation from a query is unlikely to cause a substantial increase in data transfer.

In the case of repeated database predicates, our solution is to remove the repeated predicate from the approximated query. A separate query is issued to

fetch all the relevant tuples of the repeated predicate and, once the data have been fetched from the resources, a join is performed to compute the result of the original query. The justification for this approach is that

- repeated predicates occur infrequently in practice
- by decomposing the query and caching the separate pieces, we improve the chances of obtaining future cache hits
- in the case of self-join, the operand is usually smaller than the result
- our approach does not preclude the strategy of pushing self-joins down to the resource agents. A cost model in the optimizer may still choose this as the best approach.

In general, a SQL query may contain other constructs that are outside our tractable subset, in particular GROUP BY. The GROUP BY construct itself does not affect the tractability of query containment, but, if the query contains other constructs within the scope of the GROUP BY it will also be necessary to drop the GROUP BY. However, GROUP BY is usually the last operator evaluated in a query, being applied to organize a summary of completed query results. In a heterogeneous database the common use of GROUP BY is to organize the results of the subqueries directed to individual data sources.

The performance of the cache is analyzed with respect to two major applications of InfoSleuth[9]. The EDEN application is a collaboration involving several government organizations in the United States and Europe, enabling the sharing and exchange of environmental information. TechPilot is a competitive intelligence application. TechPilot enables acquiring, integrating and monitoring technical competitive intelligence information from open sources.

To measure the overhead for the approximation scheme we need to consider, per the system's workload, the frequency in which users' queries fall outside the tractable subset, and the amount of additional data transferred as a result of weakening the queries. Neither EDEN or the competitive intelligence application contained queries outside the tractable subset. Emboldened by this result we considered the TPC-D benchmark[6]. TPC-D is a standard database benchmark comprising 17 queries intended to reflect the most complicated decision support and data mining applications for data warehouses. Only 3 of the 17 TPC-D queries were outside the subset.

**Table 2.** Result Size, in rows, for TPC-D Queries and Their Approximations

| Query Num | Result Size | Size of Approx. |
|---|---|---|
| 7F | 36 | 916 or 1444 |
| 8F | 13 | 13 |
| 16F | 322 | 1292 |

Table 2 illustrates the increase in result size induced by weakening the 3 queries. For query 8F the approximation does not increase the size of the result at all. Query 16F involves both disequation and GROUP BY, which accounts for the increase in result size. Of the three potential workloads only TPC-D query 7F yields a downside to the approach. Query 7F involves a repeated database predicate, each with low selectivity. Generalizing the repeated predicate increases the selectivity and consequently the result size. Further note that in query 7F there are two possible weakenings. The negotiation algorithm would report both possible decompositions. A cost-based optimizer could dismiss the cache mechanism entirely.

Per the performance of the cache itself. For EDEN cache hits occur in half the queries, and in the majority of those cases there is a saving of more than 50% in data transfer. For TechPilot, there were cache hits in more than half the queries, and in those cases we found a saving of more than 90% in data transfer.

## 6 Conclusions and Future Directions

An open question is whether we have defined the most general or even most useful tractable SQL subset. In other words, are there other tractable SQL subsets that may yield better coverage of real-world queries and/or tighter upper-bounds when needed? We reported favorable results, just one questionable query in three different workloads. Nevertheless, a precise way to measure these qualities has not been enunciated.

A next step is the integration of the approach to the mediator itself. A central problem solved by mediators is finding a maximally-contained rewriting of the query in terms of a fixed set of views[6]. In the terminology of knowledge compilation this corresonds to finding the smallest lower-bound cover [13].

Of general interest to the database community is how to enforce correct concurrency control based on logical predicates. It appears that our approach is directly applicable to a lock-based replication scheme proposed by Quass and Widom [15]. But general-purpose extensions are an open issue.

## References

1. C. Beeri, A. Levy and M.-C. Rousset. Rewriting queries using views in description logics. In *Proc. of ACM Symp. on Principles of Database Systems*, 1997.
2. A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proc. of the ACM Symp. on Theory of Computing*, 1977.
3. C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. In *Proc. of Int. Conf. on database theory (ICDT '97)*, pages 16–21, 1997.
4. C.M. Chen and N. Roussopoulos. The implementation and performance evaluation of the ADMS query optimizer: integrating query result caching and matching. Tech. Rep. CS-TR-3159, Computer Science Dept., University of Maryland, 1993.
5. R.W. Floyd. Algorithm 97: Shortest path. *CACM*, 5(6), June 1962.
6. H. Garcia-Molina,J. D. Ullman and J. D. Widom. *Database Systems: The Complete Book, 1/e.* Prentice-Hall, 2002.

7. A. Klug. On conjunctive queries containing inequalities. *Journal of the ACM*, 35(1), January 1988.
8. Ph.G. Kolaitis, D.L. Martin, M.N. Thakur. On the complexity of the containment problem for conjunctive queries with built-in predicates. In *Proc. of ACM Symp. on Principles of Database Systems*, 1998.
9. M. Nodine, J. Fowler, T. Ksiezyk, B. Perry, M. Taylor, A. Unruh. Active Information Gathering in InfoSleuth. IJCIS 9(1-2):3-28, 2000,
10. X. Qian. Query folding. In *Proc. of Twelfth Int. Conf. on Data Engineering*, 1996.
11. D.J. Rosenkrantz and H.B. Hunt. Processing conjunctive predicates and queries. In *Proc. of Int. Conf. on Very Large Databases*, 1980.
12. Y. Saraiya. *Subtree elimination algorithms in deductive databases.* Ph.D. thesis, Computer Science Department, Stanford University, 1991.
13. M. Schaerf and M. Cadoli. Tractable reasoning via approximation. In *Artificial Intelligence*, 74, 1995.
14. R. van der Meyden. The complexity of querying infinite data about linearly ordered domains. *Journal of Computer and System Sciences*, 54(1), 1997.
15. D. Quass and J. Widom. On-line warehouse view maintenance. In *Proc. of SIGMOD '97*, 1997.
16. X. Zhang and Z.M. Ozsoyoglu. Some results on the containment and minimization of (in)equality queries. *Information Processing Letters*, (50), 1994.

# An Algebraic Framework for Abstract Model Checking

Supratik Mukhopadhyay* and Andreas Podelski

Max-Planck-Institut für Informatik
Im Stadtwald, 66123 Saarbrücken, Germany
{supratik|podelski}@mpi-sb.mpg.de

**Abstract.** Symbolic forward analysis is a semi-algorithm that in many cases solves the model checking problem for infinite state systems in practice. This semi-algorithm is implemented in many practical model checking tools like UPPAAL [BLL+96], KRONOS [DT98] and HYTECH [HHWT97]. In most practical experiments, termination of symbolic forward analysis is achieved by employing abstractions resulting in an abstract symbolic forward analysis. This paper presents a unified algebraic framework for deriving and reasoning about abstract symbolic forward analysis procedures for a large class of infinite state systems with variables ranging over a numeric domain. The framework is obtained by lifting notions from classical algebraic theory of automata to constraints representing sets of states. Our framework provides sufficient conditions under which the derived abstract symbolic forward analysis procedure is always terminating or accurate or both. The class of infinite state systems that we consider here are (possibly non-linear) hybrid systems and (possibly non-linear) integer-valued systems. The central notions involved are those of *constraint transformer monoids* and *coverings* between constraint transformer monoids. We show concrete applications of our framework in deriving abstract symbolic forward analysis algorithms for timed automata and the two process bakery algorithm that are both terminating and accurate.

## 1 Introduction

Over the last few years, there has been an increasing research effort directed towards automatic verification of infinite state systems. Research on decidability issues (e.g., [ACJT96,ACHH93,Boi98,LPY99,HKPV95,CJ98]) has resulted in highly non-trivial algorithms for the verification of different subclasses of infinite state systems. These results do not, of course, imply termination guarantees for semi-algorithms on which practical tools are based (e.g., the decidability of the model checking problem for timed automata does not entail a termination guarantee for symbolic forward analysis of timed automata; symbolic forward analysis for timed automata is possibly non-terminating).

* Support from the grants NSF award CCR99-70925, SRC award 99-TJ-688, and DARPA ITO Mobies award F33615-00-C-1707 is gratefully acknowledged

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 152–169, 2002.

Practical tools generally use abstractions to guarantee (or speed-up) the termination of these semi-algorithms. This has led to an enormous amount of research on obtaining abstractions of infinite state systems. One line of research has been directed towards obtaining automatically or semi-automatically finite state abstractions of infinite state systems. Most of such research has been based on *predicate abstraction* [GS97,Sai00,DDP99,KN00,SS99] — a technique for automatically obtaining an abstract finite state graph from an infinite state system given a partitioning of the state space (data domain) of the system by a finite set of predicates. The finite abstract state graph can then be analyzed using a traditional finite state model checker. In [KN00], a method for automatically obtaining the predicates has also been presented. But there are two arguments against using such finite state abstractions. First, the finite state abstractions obtained tend to lose accuracy, i.e., tend to become too rough, resulting in frustrating don't know answers. Second, finiteness of the transition system is not a necessary condition for termination if an infinite state model checker is used. For example arbitrary hybrid systems can be abstracted to o-minimal [LPY99] ones (which are still infinite state) for which symbolic forward analysis is guaranteed to terminate [MP00b]. Such abstractions may result in lesser loss in accuracy than their finite state counterparts.

Another line of research has been focussed on developing abstractions such as widening that can be applied on the fly during the (infinite state) symbolic model checking. The abstract semi-algorithms resulting from such abstractions may be always terminating but approximate (i.e., they always terminate but can produce don't know answers; for example the semi-algorithm with widening used in [HPR97]), or both terminating and accurate (e.g., the algorithm with the extrapolation operator in [DT98] and used in KRONOS or the algorithm with meta-transitions described in [CJ98]) or possibly non-terminating and accurate (such abstract semi-algorithms are possibly non-terminating; but when they terminate they produce a yes/no answer; examples are the semi-algorithm with the cycle-step abstraction in [BBR97] and the semi-algorithm with accurate widening in [MP00a]). Many of these abstractions are inspired by the abstract interpretation framework of Cousot and Cousot [CC77]. A disadvantage of these types of abstractions is that there does not seem to exist any automated method for deriving them.

Symbolic forward analysis is a semi-algorithm that in many cases solves the model checking problem for infinite state systems in practice. This semi-algorithm is implemented in many practical model checking tools like UPPAAL [BLL+96], KRONOS [DT98] and HYTECH [HHWT97]. This paper presents a uniform algebraic framework for deriving abstract symbolic forward analysis procedures for a large class of infinite state systems with variables ranging over a numeric domain. We obtain the framework by lifting notions from classical algebraic theory of automata to constraints representing sets of states. Our framework provides sufficient conditions under which the derived abstract symbolic forward analysis procedure is always terminating or accurate or both (note that the derived abstraction need not be finite). The class of infinite state

systems that we consider here are (possibly non-linear) hybrid systems and (possibly non-linear) integer-valued systems. The central notions involved are those of *constraint transformer monoids* and *coverings* between constraint transformer monoids. We show concrete applications of our framework in deriving abstract symbolic forward analysis algorithms for timed automata and the two process bakery algorithm that are both terminating and accurate.

Our framework can be used to answer questions like given a finite state predicate abstraction obtained automatically by one of the methods described in [GS97,SS99,DDP99,Sai00,KN00], is it complete (i.e., does not lose any information with respect to (co)reachability)? Or given an abstraction as in [BBR97, MP00a,CJ98,DT98], is it accurate (or complete)? i.e., if the model checker gives a don't know answer is it really a don't know or it is a "no"? Or is the abstract analysis terminating? Giacobazzi, Ranzato and Scozzari [GRS00] argue that completeness (accuracy) of abstractions (abstract interpretations) is a domain property. They provide sufficient conditions on the domains under which the corresponding abstract interpretations are accurate (from now on we will use the term accurate instead of complete). They also provide methods to make abstract interpretations complete by extending or restricting the abstract domain. However their methods are directed towards program analysis and are not easily extendible to infinite state model checking. Our work presents a constraint-based algebraic framework in the context of constraint based infinite state model checking to answer questions like: given an abstract domain (and an abstraction), is the corresponding abstraction accurate?

Our results suggest a potential optimization of the (abstract) symbolic forward analysis procedures. Namely, the termination guarantees continue to hold even when the fixpoint test is made more efficient by weakening it to *local entailment* (explained below; e.g., for linear arithmetic constraints over reals, the efficiency increases from co-NP hard to polynomial).

## 2 Infinite State Systems

We use guarded-command programs to specify (possibly infinite-state) transition systems. A guarded-command program consists of a set $\mathcal{E}$ of guarded commands $e$ (called edges) of the form

$$e \;\equiv\; L = \ell \wedge \gamma_e(\boldsymbol{x}) \;[\!]\; L' = \ell' \wedge \alpha_e(\boldsymbol{x}, \boldsymbol{x'})$$

where $L$ is a variable ranging over a finite set of program locations, $\boldsymbol{x} = \langle x_1, \ldots, x_n \rangle$ is the tuple of program variables (ranging over a possible infinite numeric data domain); $\gamma_e(\boldsymbol{x})$ is a formula (the guard) whose free variables are among $\boldsymbol{x}$; $\alpha_e(\boldsymbol{x}, \boldsymbol{x'})$ is a formula (the action) whose free variables are among $\boldsymbol{x}, \boldsymbol{x'}$; $\ell$ and $\ell'$ are respectively the source and target locations of $e$. Intuitively, the primed version of a variable stands for its value in the successor state after taking a transition through a guarded command. We translate a guarded command $e$ to the logical formula $\psi_e$ simply by replacing the guard $[\!]$ with conjunction.

$$\psi_e \equiv L = \ell \wedge \gamma_e(\boldsymbol{x}) \wedge L' = \ell' \wedge \alpha_e(\boldsymbol{x}, \boldsymbol{x'})$$

A state of the system is a pair $\langle \ell, \boldsymbol{v} \rangle$ consisting of the values for the location variable and for each program variable. The state $\langle \ell, \boldsymbol{v} \rangle$ can make a transition to the state $\langle \ell', \boldsymbol{v'} \rangle$ through the edge $e$ provided that the values of $\ell$ for $L$, $\ell'$ for $L'$, $\boldsymbol{v}$ for $\boldsymbol{x}$ and $\boldsymbol{v'}$ for $\boldsymbol{x'}$ define a solution for $\psi_e$. We assume that the program variables range over the set of natural numbers $\mathcal{N}$ or the set of reals $\mathcal{R}$, and the guard and the action formulas are $Arith(\mathcal{N})$ (the theory of natural numbers with addition, multiplication and order; it is interpreted over the structure $\langle \mathcal{N}, <, +, \cdot, 0, 1 \rangle$) or $OF(\mathcal{R})$ (the theory of the ordered field of reals; it is interpreted over the structure $\langle \mathcal{R}, <, +, \cdot, 0, 1 \rangle$) formulas. Below, we will refer to $OF(\mathcal{R})$ or $Arith(\mathcal{N})$ formulas as constraints. For a formula $\varphi$ with free variables $\boldsymbol{x}$, we denote by $\varphi(\boldsymbol{x'})$, the formula obtained by replacing the free variables $\boldsymbol{x}$ of $\varphi$ by $\boldsymbol{x'}$. We will use constraints $\varphi$ to represent certain sets of states of the system. We identify solutions of the constraints with the states of the system. We write $\langle \ell, \boldsymbol{v} \rangle \models \varphi$ to denote that the state that the state $\langle \ell, \boldsymbol{v} \rangle$ is a solution to the constraint $\varphi$. For a constraint $\varphi$, we define the denotation of $\varphi$, denoted by $[\varphi]$, as

$$[\varphi] = \{\langle \ell, \boldsymbol{v} \rangle \mid \langle \ell, \boldsymbol{v} \rangle \models \varphi\}.$$

For any two constraints $\varphi$ and $\varphi'$, we write $\varphi \models \varphi'$ ($\varphi$ entails $\varphi'$) iff $[\varphi] \subseteq [\varphi']$. We also assume that, for the class of constraint systems that we are dealing with, it is decidable for any two constraints $\varphi$ and $\varphi'$, whether $\varphi \models \varphi'$. We single out a constraint $\varphi^0$ as the initial constraint. In the sequel, we assume only *conjunctive* constraints; i.e., constraints that are conjunctions of atomic constraints of the form $t \; relop \; c$ where $t$ is a term, $c \in \mathcal{N}$ and $relop \in \{>, <, \geq, \leq\}$. Examples of systems as described above include the bakery algorithm, the bounded buffer producer-consumer problem etc. as well as the so-called hybrid systems.

## 3 Constraint Transformer Monoids

Our definition of constraint transformer monoids is inspired by the definition of (syntactic) transformation monoids in [Eil76]. Let $\Phi$ be a (possibly infinite) set of satisfiable constraints (i.e., each constraint in $\Phi$ is satisfiable). We denote the set of all partial functions $\Phi \longrightarrow \Phi$ by $\mathcal{SF}(\Phi)$. Let $1_\Phi$ denote the identity function. The set $\mathcal{SF}(\Phi)$ forms a monoid with functional composition as the multiplication and $1_\Phi$ as the identity element. A *constraint transformer semigroup* is a pair $\langle \Phi, S \rangle$ where $S$ is a subsemigroup of $\mathcal{SF}(\Phi)$. The constraint transformer semigroup $\langle \Phi, S \rangle$ is a constraint transformer monoid if the identity function $1_\Phi$ is in $S$. The elements of $\Phi$ are called *symbolic states*. The elements of $S$ are called *constraint transformers*. A constraint transformer monoid $X = \langle \Phi, S \rangle$ is a constraint transformer submonoid of a constraint transformer $Y = \langle \Phi', S' \rangle$ if $\Phi \subseteq \Phi'$ and $S$ is a submonoid of $S'$.

By the denotation of set of constraints $\Phi$, we represent the denotation of their disjunction; i.e., $[\Phi] = \bigcup_{\varphi \in \Phi} [\varphi]$.

We next define a *syntactic order* $\sqsubseteq^\varphi$ on a constraint transformer monoid $X = \langle \Phi, S \rangle$ with respect to a constraint $\varphi \in \Phi$ as follows.

**Syntactic Order.** For $w, w' \in S$, $w \sqsubseteq^{\varphi} w'$ iff $w(\varphi) \models w'(\varphi)$. Given a set of constraints $\Psi \subseteq \Phi$, we say that an infinite sequence $w_0, w_1, \ldots$, where $w_i \in S$, is *syntactically increasing* with respect to $\Psi$ if for all $i \geq 1$, there exists $\varphi \in \Psi$ such that $w_i \not\sqsubseteq^{\varphi} w_j$ for all $j < i$.

**Finitary Constraint Transformer Monoids.** We say that a constraint transformer monoid $X$ is *finitary* with respect to a set $\Psi \subseteq \Phi$ of constraints if there does not exist any syntactically increasing infinite sequence with respect to $\Psi$. Note that $X$ is finitary does not mean that $\Phi$ is finite.

*Reachability.* For a constraint transformer monoid $X = \langle \Phi, S \rangle$, a reachability question is of the form: given $\varphi^1, \varphi^2 \in \Phi$, does there exist a $w \in S$ such that $\varphi^2 = w(\varphi^1)$?

*Constraint transformer monoids generated by infinite state systems:* We now show how an infinite state system generates a constraint transformer monoid. We write $e_1, \ldots, e_m$ for the word $w$ obtained by concatenating the 'letters' $e_1, \ldots, e_m$ (where each $e_i$ is an edge of a guarded command program); thus, $w$ is a word over the set of edges (or guarded commands) $\mathcal{E}$, i.e., $w \in \mathcal{E}^*$ (assuming that the target location of $e_i$ is the source location of $e_{i+1}$). *We identify two constraints $\varphi$ and $\varphi'$ iff they have the same denotations; i.e.,* $[\varphi] = [\varphi']$. We define the constraint transformer with respect to an edge $e$ as the successor constraint function $[\![e]\!]$ that assigns to a constraint $\varphi$ the constraint

$$[\![e]\!](\varphi) \equiv ((\exists \boldsymbol{x}(\varphi \wedge \psi_e))(\boldsymbol{x})).$$

The successor constraint function $[\![w]\!]$ with respect to a string $w = e_1 \ldots e_m$ of length $m \geq 0$ is the functional composition of the functions with respect to the edges $e_1, \ldots, e_m$, i.e., $[\![w]\!] = [\![e_1]\!] \circ \ldots \circ [\![e_m]\!]$. Thus $[\![\varepsilon]\!](\varphi) = \varphi$ and $[\![w.e]\!](\varphi) = [\![e]\!]([\![w]\!](\varphi))$. The solutions of $[\![w]\!](\varphi)$ are exactly the successors of the solutions of $\varphi$ obtained by taking the sequence of transitions through the guarded commands $e_1, \ldots, e_m$ (in that order). The constraint transformer monoid generated by an infinite state system $\mathcal{S}$ is given by $CT(\mathcal{S}) = \langle \Phi, S \rangle$ where $\Phi = \{\varphi \mid \exists w \in \mathcal{E}^* [\![w]\!](\varphi^0) = \varphi\}$ and $S = \{[\![w]\!] \mid w \in \mathcal{E}^*\}$ with functional composition as the multiplication in $S$ and $[\![\varepsilon]\!]$ as the unit element.

## 4 Coverings of Constraint Transformer Monoids

Our definition of covering between constraint transformer monoids is inspired by that of covering between (syntactic) transformer monoids in [Eil76]. Let $X = \langle \Phi, S \rangle$ and $Y = \langle \Phi', S' \rangle$ be two constraint transformer monoids. Let $f$ be a total (binary) relation from $\Phi$ to $\Phi'$. For $w \in S$ and $v \in S'$, we consider the following diagram.

$$\begin{array}{ccc} \Phi & \xrightarrow{w} & \Phi \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle f} \\ \Phi' & \xrightarrow[v]{} & \Phi' \end{array}$$

If the above diagram commutes, i.e., for all $\varphi \in \Phi$, $v(\{\psi \mid f(\varphi,\psi)\}) = \{\psi \mid f(w(\varphi),\psi)\}$, then we say that $v$ *covers* $w$ with respect to $f$ where for a set $\widetilde{\Phi}$, $v(\widetilde{\Phi}) = \{v(\varphi) \mid \varphi \in \widetilde{\Phi}\}$. If for each $w \in S$ there exists a $v \in S'$ such that $v$ covers $w$ we say that the relation $f$ is a *covering* between $X$ and $Y$. We say that the constraint transformer monoid $Y$ covers the constraint transformer monoid $X$ if a covering $f$ exists between $X$ and $Y$ and we write $X \prec Y$. We are now going to define a quotient of $X$ with respect to $f$; we call such a quotient an $f$-quotient of $X$.

**$f$ quotient.** In order to define an $f$-quotient of $X$, we first define an equivalence relation $\sim_f$ on $\Phi$ as follows. For $\varphi, \varphi' \in \Phi$,

$$\varphi \sim_f \varphi' \iff \{\psi \mid f(\varphi,\psi)\} = \{\psi' \mid f(\varphi',\psi')\}.$$

Next we define a representant function $rep : \Phi/\sim_f \longrightarrow \Phi$ as $rep([\varphi]) = \varphi$, where $[\varphi]$ is the equivalence class of $\varphi$ with respect to the equivalence relation $\sim_f$. Given a covering relation $f$ and a representant function $rep$ as above, we call the constraint transformer monoid $X' = \langle rep(\Phi/\sim_f), \hat{S}\rangle$ an $f$-quotient of $X$ where $\hat{S} = \{\widetilde{w} \mid w \in S\}$ and for any constraint $\psi \in rep(\Phi/\sim_f)$, $\widetilde{w}(\psi) = rep([\psi'])$ iff $w(\psi) = \psi'$.

**Canonicity and Saturation.** We say that a constraint $\varphi \in \Phi$ is *canonical* with respect to $f$ if for all $\varphi' \in \Phi$ with $\varphi \not= \varphi'$, $\{\psi \mid f(\varphi,\psi)\} \neq \{\psi' \mid f(\varphi',\psi')\}$. We say that the relation $f$ *saturates* a constraint $\varphi \in \Phi$ if there exists a constraint $\psi \in \Phi'$ such that $f(\varphi,\psi)$ and for all $\varphi' \in \Phi'$ with $f(\varphi',\psi)$, we have $\varphi = \varphi'$. The notions of canonicity and saturation indicate the "local" distinguishing power of $f$.

**Definition 1 (Homeocovering).** *We say that $f$ is a* homeocovering *from $X$ to $Y$ with respect to constraints $\varphi^1$ and $\varphi^2$ if $f$ is a covering from $X$ to $Y$ and one of the following conditions hold.*

- *either $f^{-1}$ is a covering from $Y$ to a constraint transformer submonoid $X' = \langle \Phi'', S''\rangle$ of $X$ (i.e., $Y \prec X'$ and $f^{-1}$ witnesses the covering) and $\varphi^1, \varphi^2 \in \Phi''$,*
- *or $f^{-1}$ is a covering from $Y$ to an $f$-quotient $X'$ of $X$ (i.e., $Y \prec X'$ and $f^{-1}$ is a witness to this covering) and $\varphi^1$ and $\varphi^2$ are both canonical with respect to $f$*

**Definition 2 (Finitary Covering).** *We say that a covering $f$ is a* finitary covering *from $X$ to $Y$ with respect to a set of constraints $\Psi \subseteq \Phi$, if $f$ is a covering from $X$ to $Y$ and $Y$ is finitary with respect to $\{\psi \mid f(\varphi,\psi), \varphi \in \Psi\}$.*

Note that even if $f$ is a finitary covering from $X$ to $Y = \langle \Phi', S'\rangle$, it does not mean that $\Phi'$ is finite. We will use the notion of finitary coverings to provide sufficient conditions for termination of abstract symbolic forward analysis in Theorem 1.

**Proposition 1.** *Let $\mathcal{S}$ be an infinite state system. Let $X = \langle \Phi, S \rangle$ be the constraint transformer monoid generated by $\mathcal{S}$. Let $Y = \langle \Phi', S' \rangle$ be a constraint transformer such that $X \prec Y$ with $f$ being a covering between $X$ and $Y$. Suppose that a constraint $\varphi^2$ is reachable from the initial constraint $\varphi^1$ in $\mathcal{S}$. Then there exists $v \in S'$ such that*

$$\{\psi \mid f(\varphi^2, \psi)\} = v(\{\psi^1 \mid f(\varphi^1, \psi^1)\}).$$

*If, in addition, $f$ saturates $\varphi^1$ and $f$ is homeocovering from $X$ to $Y$ with respect to $\varphi^1$ and $\varphi^2$, then the converse also holds.*

*Proof.* See the Appendix. ∎

## 5 Constraint Trees and Symbolic Forward Analysis

Given a constraint transformer monoid $X = \langle \Phi, S \rangle$ with a finite set of generators $\widetilde{S}$ (i.e., $\widetilde{S}$ generates $S$), we define the constraint tree for $X$ as follows. Let $S^{free}$ be the free monoid generated by $\widetilde{S}$. For $\widetilde{w} \in S^{free}$, we say that $w \in S$ is the *companion* of $\widetilde{w}$ iff $w$ is obtained by replacing concatenation in $\widetilde{w}$ with multiplication in $S$. Thus, for example, $w \in S$ is a companion of $g_1.g_2$ iff $w = g_1 \circ g_2$ where $\circ$ is the multiplication in $S$.

**Definition 3 (Constraint Tree).** *The constraint tree for $X = \langle \Phi, S \rangle$ with respect to a constraint $\varphi^0 \in \Phi$ and a finite set of generators $\widetilde{S}$ of $S$ is an infinite tree with domain $S^{free}$ that labels the node $\widetilde{w}$ by the constraint $w(\varphi^0)$ where $w$ is the companion of $\widetilde{w}$.*

That is, the root $\varepsilon$ is labeled with $\varphi^0$. For a node $\widetilde{w}$ labeled $\varphi$, for each $g \in \widetilde{S}$, the successor node $\widetilde{w}.g$ is labeled by $g(\varphi)$. We are now in a position to define symbolic forward analysis of a finitely generated constraint transformer monoid with respect to a constraint formally. A symbolic forward analysis is a traversal of (a finite prefix of) a constraint tree in a particular order. The following definition of a non-deterministic procedure abstracts away from that specific order.

**Definition 4 (Symbolic Forward Analysis).** *A symbolic forward analysis of a finitely generated constraint transformer monoid $X$ with respect to a constraint $\varphi^0$ and a finite set of generators $\widetilde{S}$ is a procedure that enumerates constraints $\varphi_i$ labeling the nodes $\widetilde{w}_i$ of the constraint tree of $X$ with respect to $\varphi^0$ and $\widetilde{S}$ in a tree order such that the following holds.*

- *$\varphi_i = w_i(\varphi^0)$ for $0 \leq i < B$ where the bound $B$ is either a natural number or $\omega$ and $w_i$ is the companion of the word $\widetilde{w}_i \in S^{free}$,*
- *if $\widetilde{w}_i$ is a prefix of $\widetilde{w}_j$ then $i \leq j$,*
- *the disjunction $\bigvee_{0 \leq i < B} \varphi_i$ is equivalent to the disjunction $\bigvee_{0 \leq i < \omega} \varphi_i$.*

The number $i$ is a leaf of a symbolic forward analysis if the node $\widetilde{w}_i$ is a leaf of the tree formed by all the nodes $\widetilde{w}_i$ where $0 \leq i \leq B$. We say that a symbolic forward analysis terminates if its bound $B$ is finite. We define that a symbolic forward analysis terminates with local entailment if for all its leaves $i$ there exists a $j < i$

such that the constraint $\varphi_i$ entails the constraint $\varphi_j$ (remember that each $\varphi_i$ is a conjunctive constraint). In contrast, a symbolic forward analysis terminates with global entailment if for all its leaves $i$, the constraint $\varphi_i$ entails the disjunction of the constraints $\varphi_j$ where $j < i$. For constraint domains that do not satisfy the independence property[1], checking for global entailment is usually more expensive than checking for local entailment. Many model checkers use local entailment for their fixpont test (e.g., UPPAAL [LPY95] uses identity; the model checker for infinite state systems described in [DP99] uses local entailment).

*Remark 1.* A symbolic forward analysis for an infinite state system $\mathcal{S}$ with respect to the initial constraint $\varphi^0$ is a symbolic forward analysis of the constraint transformer monoid generated by $\mathcal{S}$ with respect to $\varphi^0$. If terminating, the constraint $\bigvee_{0 \leq i < B} \varphi_i$ represents the set of all reachable states in $\mathcal{S}$. For an infinite state system $\mathcal{S}$, a constraint $\varphi^2$ is reachable from the constraint $\varphi^1$ if there exists a node $\widetilde{w}$ labeled by $\varphi^2$ in the constraint tree with respect to $\varphi^1$ of the constraint transformer monoid generated by $\mathcal{S}$.

## 6 Abstract Constraint Trees and Abstract Symbolic Forward Analysis

Let $X = \langle \Phi, S \rangle$ be a constraint transformer monoid with a finite set of generators $\widetilde{S}$. Let $S^{free}$ be the free monoid generated by $\widetilde{S}$. Let $Y = \langle \Phi', S' \rangle$ be a constraint transformer monoid covering $X$ and let $f$ be a covering relation witnessing the covering. We define an abstract constraint tree of $\mathcal{S}$ with respect to $Y$, $f$, $\widetilde{S}$ and a constraint $\varphi^0$ as follows.

**Definition 5 (Abstract Constraint Tree).** *An abstract constraint tree for $X$ with respect to the constraint transformer monoid $Y$, a constraint $\varphi^0$, a finite set of generators $\widetilde{S}$ and a covering relation $f$ is an infinite tree with domain $S^{free}$ that labels the node $\widetilde{w} \in S^{free}$ by the set of constraints $\Psi = \{v(\psi^0) \mid f(\varphi^0, \psi^0)\}$ where $v \in S'$ covers $w$ (the companion of $\widetilde{w}$).*

In the above definition we assume that there is a *finite representation* for each $\Psi$ labeling $\widetilde{w} \in S^{free}$ in the abstract constraint tree. Note that the constraint tree for $X$ with respect to $\varphi^0$ is an abstract constraint tree for $X$ with respect to the constraint transformer monoid $X$ and the identity function as the covering. Also note that the constraint transformer monoid $Y$ may be arbitrary; i.e., it need not be finitely generated. If for each $w \in S$, we fix a $v \in S'$ covering $w$, we call the resulting abstract constraint tree a fixed-cover abstract constraint tree. Below, whenever we talk about abstract constraint tree, we assume a fixed cover $\mathcal{C} \subseteq S'$, i.e., for each $w \in S$ there exists a unique $v \in \mathcal{C}$ such that $v$ covers $w$. We denote by $T_{\mathcal{C}}$ be the abstract constraint tree of $X$ with respect to $Y$, $f$ and $\mathcal{C}$, i.e., a node $\widetilde{w}$ is labeled by $\{v(\psi^0) \mid f(\varphi^0, \psi^0)$ where $v$ is the unique element of $\mathcal{C}$ covering $w$ (the companion of $\widetilde{w}$) and $\{\psi^0 \mid f(\varphi^0, \psi^0)\}$ labels the root. We

[1] A constraint domain is said to satisfy the independence property if for any constraint $\psi$ and a set of constraints $\Phi$, $\psi \models \bigvee_{\varphi \in \Phi} \varphi$ iff there exists $\varphi \in \Phi$ such that $\psi \models \varphi$

are now in a position to define formally abstract symbolic forward analysis. An abstract symbolic forward analysis of $X$ with respect to a constraint transformer monoid $Y$ is a traversal of (a finite prefix of) the (fixed cover) abstract constraint tree of $X$ with respect to $Y$ in a particular order. The following definition of a non-deterministic procedure abstracts away from that specific order.

**Definition 6 (Abstract Symbolic Forward Analysis).** *An abstract symbolic forward analysis of a constraint transformer monoid $X$ with respect to a constraint $\varphi^0$ and a fixed cover $\mathcal{C}$ is a procedure that enumerates the sets of constraints $\Psi_i$ labeling the nodes $\widetilde{w_i}$ of the abstract constraint tree $T_{\mathcal{C}}$ with respect to $\varphi^0$ (and $\mathcal{C}$) in a tree order such that the following holds.*

- $\Psi_i = \{v_i(\psi^0) \mid f(\varphi^0, \psi^0)\}$ *where* $v_i \in \mathcal{C}$ *covers* $w_i \in S$ *(the companion of* $\widetilde{w_i}$*) and the bound $B$ is either a natural number or $\omega$,*
- *if* $\widetilde{w_i}$ *is a prefix of* $\widetilde{w_j}$ *then* $i \leq j$,
- *the disjunction* $\bigvee_{0 \leq i < B} \bigvee \Psi_i$ *is equivalent to the disjunction* $\bigvee_{0 \leq i < \omega} \bigvee \Psi_i$ *where* $\bigvee \Psi_i \equiv \bigvee_{\varphi \in \Psi_i} \varphi$.

Similar to symbolic forward analysis, we say that an abstract symbolic forward analysis terminates if the bound $B$ is finite; the concept of a leaf is defined similarly. We say that an abstract symbolic forward analysis terminates with local entailment if for all its leaves $i$, for each constraint $\varphi \in \Psi_i$, there exists a $j < i$, and a constraint $\varphi' \in \Psi_j$ such that $\varphi \models \varphi'$. The notion of termination with global entailment is defined in the obvious way.

We now present sufficient conditions under which an abstract symbolic forward analysis is possibly non-terminating and accurate, terminating and possibly inaccurate or both terminating and accurate with respect to a reachability question.

**Theorem 1.** *Let $X = \langle \Phi, S \rangle$ be a constraint transformer monoid having a finite set of generators $\widetilde{S}$. Let $\varphi^1, \varphi^2 \in \Phi$. Let $Y = \langle \Phi', S' \rangle$ be a constraint transformer monoid covering $X$ with $f$ witnessing the covering and let $\mathcal{C} \subseteq S'$ be a fixed cover. Then the following hold.*

1. *Suppose that for all $i$, $\Psi_i \not= \{\psi \mid f(\varphi^2, \psi)\}$ where $\Psi_i$ is the set of constraints labeling the node $\widetilde{w_i}$ of the abstract constraint tree of $X$ with respect to $\varphi^1$, $Y$, $f$, and $\mathcal{C}$. Then the constraint $\varphi^2$ is not reachable from $\varphi^1$ in $X$.*
   a) *If, in addition, $f$ is a finitary covering with respect to $\{\varphi^1\}$, then each abstract symbolic forward analysis of $X$ with respect to $\varphi^1$, $Y$, $f$ and $\mathcal{C}$ terminates with local entailment. In this case, abstract symbolic forward analysis always terminates with local entailment but may produce a 'don't know' answer the reachability question.*
2. *If $f$ saturates $\varphi^1$ and $f$ is a homeocovering from $X$ to $Y$ with respect to constraints $\varphi^1$ and $\varphi^2$ then $\varphi^2$ is reachable from $\varphi^1$ in $X$ iff there exists an $i$ such that $\Psi_i = \{\psi \mid f(\varphi^2, \psi)\}$ where $\Psi_i$ labels the node $\widetilde{w_i}$ in the abstract constraint tree of $X$ with respect to $Y$, $f$, $\varphi^1$ and $\mathcal{C}$. In this case, abstract symbolic forward analysis is possibly non-terminating; but when it terminates, it produces a yes/no answer for the reachability question.*

*a) In particular, if $f$ is a function then $\varphi^2$ is reachable from $\varphi^1$ in $X$ iff $\psi$ is not reachable from $\psi^0$ in the abstract symbolic forward analysis of $X$ with respect to $Y$, $f$, $\varphi^1$ and $\mathcal{C}$ where $f(\varphi^1, \psi^0)$ and $f(\varphi^2, \psi)$.*

*b) If, in addition, $f$ is a finitary covering with respect to $\{\varphi^1\}$ then each abstract symbolic forward analysis of $X$ with respect to $\varphi^1$, $Y$, $f$ and $\mathcal{C}$ terminates with local entailment. In this case, abstract symbolic forward analysis always terminates with local entailment and is accurate.*

*Proof.* See the Appendix. ∎

## 7 Applications

In this section, we show concrete applications of the framework developed above to timed automata and the two-process bakery algorithm.

### 7.1 Timed Automata

A *timed automaton* $\mathcal{U}$ can, for the purpose of reachability analysis, be defined as a set $\mathcal{E}$ of guarded commands (called edges) of the form

$$e \;\equiv\; L = \ell \wedge \gamma_e(\boldsymbol{x}) \;[\!]\; L' = \ell' \wedge \alpha_e(\boldsymbol{x}, \boldsymbol{x'}).$$

Here $L$ is a variable ranging over the finite set of *locations*, and $\boldsymbol{x} = \langle x_1, \ldots, x_n \rangle$ are the variables standing for the clocks and ranging over nonnegative real numbers. As usual, the primed version of a variable stands for its value after the transition.

The guard formula $\gamma_e(\boldsymbol{x})$ over the variables $\boldsymbol{x}$ is built up from conjuncts of the form $x_i \sim k$ where $x_i$ is a clock variable, $\sim$ is a comparison operator (i.e., $\sim \in \{=, <, \leq, >, \geq\}$) and $k$ is a natural number.

The action formula $\alpha_e(\boldsymbol{x}, \boldsymbol{x'})$ of $e$ is defined by a subset $\mathrm{Reset}_e$ of $\{1, \ldots, n\}$ (denoting the clocks that are *reset*); it is of the form

$$\alpha_e(\boldsymbol{x}, \boldsymbol{x'}) \;\equiv\; \exists z \geq 0 \bigwedge_{i \in \mathrm{Reset}_e} x'_i = z \wedge \bigwedge_{i \notin \mathrm{Reset}_e} x'_i = x_i + z.$$

The existentially quantified variable $z$ in the action formula denotes delay of time. We write $\psi_e$ for the logical formula corresponding to $e$ (with the free variables $\boldsymbol{x}$ and $\boldsymbol{x'}$; we replace the guard symbol $[\!]$ with conjunction).

$$\psi_e(\boldsymbol{x}, \boldsymbol{x'}) \;\equiv\; L = \ell \wedge \gamma_e(\boldsymbol{x}) \wedge L' = \ell' \wedge\; \alpha_e(\boldsymbol{x}, \boldsymbol{x'})$$

The states of $\mathcal{U}$ (called *positions*) are tuples of the form $\langle \ell, \boldsymbol{v} \rangle$ consisting of values for the location and for each clock. The position $\langle \ell, \boldsymbol{v} \rangle$ can make an *edge transition* using $e$ in combination with a *time transition* to the position $\langle \ell', \boldsymbol{v'} \rangle$ if the values $\ell$ for $L$, $\ell'$ for $L'$ etc. define a solution for $\psi_e$. The position $\langle \ell, \boldsymbol{v} \rangle$ can make a *time transition* to any position $\langle \ell, \boldsymbol{v} + \delta \rangle$ where $\delta \geq 0$ is a real number.

(An edge transition by itself is defined if we replace the variable $z$ in the formula for $\alpha$ by the value 0.).

Symbolic forward analysis of timed automata is possibly non-terminating [MP99]. In order to define an abstract symbolic forward analysis for timed automata, we need the following operation, called 'trim', on constraints. At a high level, the trim operation can be viewed as a constraint manipulation operation. The basic intuition is as follows: once the value of a clock goes above the maximal constant $M$ occurring in the constraints of a timed automaton, it does not matter what the value is. All such values are similar. Hence, if a constraint has a solution in which the value of a variable is above the maximal constant $M$, then the constraint can be manipulated to incorporate all "similar tuples'. In the definition below, we assume that constraints are in a normalized form. It can be shown that for timed automata the reachable constraints can be represented in a normalized form and there exists an algorithm for doing it in polynomial time.

**Definition 7 (Trim).** *We define an operator trim, which given a satisfiable constraint $\varphi$, produces a constraint $\varphi' = trim(\varphi)$, by the method given below. The constraint $trim(\varphi)$ is obtained from the normalized form of $\varphi$ by the following operations:*

- *Remove all constraints of the form $x_j - x_i > a$ or $x_j - x_i \geq a$, for each pair of variables $x_i, x_j$, $i \neq j$, such that $\varphi \wedge x_i > M$ is satisfiable and $\exists_{-x_j}(\varphi)$ is equivalent to $\exists_{-x_j}(\varphi \wedge x_i > M)$ and $(\varphi \wedge x_j > M)$ is not equivalent to $\varphi$, where $a$ is an integer and the existential quantifier is over all variables but $x_j$.*
- *Remove all constraints of the form $x_i < c$ or $x_i \leq c$ where $c$ is an integer and $c > M$.*
- *For each $i$, such that $(\varphi \wedge x_i > M)$ is equivalent to $\varphi$, replace all the constraints of the form $x_i - x_j \sim a$ or $x_i \sim c$ by the constraint $x_i > M$, where $a$ and $c$ are integers and $c > M$ and $\sim \in \{>, \geq\}$.*

Let $\mathcal{T}$ be a timed automaton and let $X = \langle \Phi, S \rangle$ be the constraint transformer monoid generated by $\mathcal{T}$. We define the constraint transformer monoid $Y$ obtained by trimming as follows.

**Definition 8 (Constraint Transformer Monoid obtained by trimming).** *Given a timed automaton $\mathcal{T}$, the constraint transformer monoid $Y = \langle \Phi', S' \rangle$, where $\Phi' = \{trim(\varphi) \mid \varphi \in \Phi\}$ and $S' = \{\widetilde{w} \mid w \in S\}$ and $\widetilde{w}(trim(\varphi)) = trim(\varphi')$ if $w(\varphi) = \varphi'$.*

It can be easily verified that each $\widetilde{w}$ is a function from $\Phi'$ to $\Phi'$ and that $S'$ is a monoid with the identity function as the unit element.

**Proposition 2.** *For a timed automaton $\mathcal{T}$ with the generated constraint transformer monoid $X = \langle \Phi, S \rangle$, the constraint transformer monoid $Y$ obtained by trimming covers $X$ with the function $f : \varphi \mapsto trim(\varphi)$ (note that the trim operation is a function) witnessing the covering.*

Intuitively, each $w$ is covered with respect to $f$ by $\widetilde{w}$.

**Proposition 3.** *The constraint transformer monoid $Y$ obtained by trimming is finite.*

*Proof.* See the Appendix. ∎

**Proposition 4.** *Any $f$-quotient of $X'$ of $X$ covers the constraint transformer monoid $Y$ obtained by trimming with $f^{-1}$ witnessing the covering.*

We call a constraint $\varphi$ *bounded* if $\varphi \wedge \bigwedge_{i=1}^{n} x_i \leq M = \varphi$. It can be easily verified that any bounded constraint $\varphi \in \Phi$ is canonical with respect to $f$. Also for each bounded constraint $\varphi \in \Phi$, $f$ saturates $\varphi$.

**Theorem 2.** *For any constraint $\varphi$, abstract symbolic forward analysis of a timed automaton $\mathcal{T}$ with respect to the constraint transformer monoid $Y$ obtained by trimming, $f$ and $\varphi$ terminates. Moreover, if $\varphi, \varphi'$ is are bounded constraints, then $\varphi'$ is reachable from $\varphi$ in $\mathcal{T}$ iff $f(\varphi')$ is reachable in the abstract symbolic forward analysis of $\mathcal{T}$ with respect $Y$, $f$ and $\varphi$.*

*Proof.* Follows from Propositions 1, 2, 3, 4 and Theorem 1. ∎

Note that the constraint transformer monoid $Y$ above is never constructed explicitly. Rather, it is constructed on-the-fly.

### 7.2 The Two-Process Bakery Algorithm

The bakery algorithm implements a mutual exclusion protocol. The guarded commands for the two-process bakery algorithm are given in figure 1. We say that the two process bakery algorithm is safe if no state of the form $L = \langle use, use \rangle \wedge \psi$ is reachable from the initial state. Let $X = \langle \Phi, S \rangle$ be the constraint transformer monoid generated by the two-process bakery algorithm. We define the covering monoid called the *abstract target monoid* as follows.

**Definition 9 (Abstract Target Monoid).** *Given the two process bakery algorithm, the abstract target monoid $Y$ is defined as $Y = \langle \Phi', S' \rangle$ where $\Phi' = \{\varphi_1, \ldots, \varphi_{10}\}$ and $S' = \{\widetilde{w} \mid [\![w]\!] \in S\}$ where the constraints $\varphi_1, \ldots, \varphi_{10}$ are defined in Figure 2.*

The constraints in Figure 2 are obtained by a simple inspection of the guards and the actions of the composed transition system but a similar abstraction could have been obtained automatically using one of the methods presented in [Sai00, SS99,GS97,KN00,DDP99]. Here $\widetilde{w}(\varphi_i) = \varphi_j$ if there exists $\psi, \psi' \in \Phi$ such that $[\![w]\!](\psi) = \psi'$ and $\psi \models \varphi_i$ and $\psi' \models \varphi_j$. It can be easily verified that each $\widetilde{w} \in S'$ is a function from $\Phi'$ to $\Phi'$. Define the relation $f$ from $\Phi$ to $\Phi'$ as $f(\varphi, \varphi')$ iff $\varphi \models \varphi'$. Note that $f$ is a function in this case.

**Proposition 5.** *The abstract target monoid $Y$ covers the constraint transformer monoid $X$ (generated by the two-process bakery algorithm) with the mapping $f$ witnessing the covering.*

*Proof.* Follows from the definitions of $S'$ and $f$. ∎

Each $w$ is covered with respect to $f$ by $\widetilde{w}$.

**Proposition 6.** *Any $f$-quotient of $X$ covers the abstract target monoid $Y$ with $f^{-1}$ witnessing the covering.*

**Control variables:** $p_1$, $p_2$ varying on $\{think, wait, use\}$
**Data variables:** $a_1$, $a_2 \geq 0$.
**Initial condition:** $p_1 = think \wedge p_2 = think \wedge a_1 = a_2 = 0$
**Transitions for** $i, j : 1, 2$, $i \neq j$**:**

$$\begin{array}{lll} \tau_{t_i} : : & p_i = think & \| \; p_i' = wait \wedge a_i' = a_j + 1 \\ \tau_{w_i} : : & p_i = wait \wedge a_i < a_j & \| \; p_i' = use \\ \tau_{w_i'} : : & p_i = wait \wedge a_j = 0 & \| \; p_i' = use \\ \tau_{u_i} : : & p_i = use & \| \; p_i' = wait \wedge a_i' = 0 \end{array}$$

**Fig. 1.** The bakery algorithm

**Theorem 3.** *For any constraint $\varphi$, any abstract symbolic forward analysis of the two-process bakery algorithm with respect to $\varphi$, the abstract target monoid $Y$ and $f$ terminates. Moreover, for any two constraints $\varphi$ and $\varphi'$ such that $f$ saturates both $\varphi$ and $\varphi'$, $\varphi'$ is reachable from $\varphi$, iff $\varphi_j$, such that $f(\varphi', \varphi_j)$, is reachable from $\varphi_i$, such that $f(\varphi, \varphi_i)$, in an abstract symbolic forward analysis wrt $\varphi$, $Y$ and $f$. In particular, the two-process bakery algorithm is safe iff the constraint $L = \langle use, use\rangle \wedge a_1 \geq 0 \wedge a_2 \geq 0$ is reachable in the abstract symbolic forward analysis with respect to $L = \langle think, think\rangle \wedge a_1 = 0 \wedge a_2 = 0$, $f$ and the abstract target monoid $Y$.*

*Proof.* Follows from Propositions 1, 5, 6 and Theorem 1. ‖

$$\begin{array}{ll} \varphi_1 & \equiv L = \langle think, think\rangle \wedge a_1 = 0 \wedge a_2 = 0 \\ \varphi_2 & \equiv L = \langle wait, think\rangle \wedge a_1 \geq 0 \wedge a_2 = 0 \\ \varphi_3 & \equiv L = \langle think, use\rangle \wedge a_1 = 0 \wedge a_2 \geq 0 \\ \varphi_4 & \equiv L = \langle use, think\rangle \wedge a_1 \geq 0 \wedge a_2 = 0 \\ \varphi_5 & \equiv L = \langle wait, wait\rangle \wedge a_1 = a_2 + 1 \wedge a_2 \geq 1 \\ \varphi_6 & \equiv L = \langle wait, wait\rangle \wedge a_2 = a_1 + 1 \wedge a_1 \geq 1 \\ \varphi_7 & \equiv L = \langle use, wait\rangle \wedge a_2 = a_1 + 1 \wedge a_2 \geq 1 \\ \varphi_8 & \equiv L = \langle think, wait\rangle \wedge a_1 = 0 \wedge a_2 \geq 0 \\ \varphi_9 & \equiv L = \langle wait, use\rangle \wedge a_1 \geq 1 \wedge a_1 = a_2 + 1 \\ \varphi_{10} & \equiv L = \langle use, use\rangle \wedge a_1 \geq 0 \wedge a_2 \geq 0 \end{array}$$

**Fig. 2.** Constraints in $\Phi'$

## 8 Summary and Related Work

We have presented a new algebraic theory for abstract symbolic forward analysis. Our framework is well suited to constraint based symbolic model checking of infinite state systems. Our framework provides sufficient conditions under which

the abstract symbolic forward analysis is always terminating or accurate or both. Note that the covering constraint transformer monoid can be arbitrary (i.e., may not be finitely generated). Also note that the sufficient termination conditions in our framework do not require the covering constraint transformer to be finite. Also the termination guarantees continue to hold even when the fixpoint test is weakened to local entailment.

We have already commented about the finite state abstraction strategies [GS97,SS99,Sai00,DDP99,KN00] in the Introduction. Our framework can be used as a mathematical basis for predicate abstraction a' la' [GS97,Sai00]. Our framework can be incorporated in these methodologies to increase their usefulness. In [CGJ$^+$00], Clarke et. al. present a methodology for counter example guided abstraction refinement. If the model checker produces a don't know answer (i.e., a spurious counter example), their method uses the spurious counter example to refine the abstraction. However, their method is directed to finite state systems. It is not clear how to extend their method for infinite state systems. In [AAB00], Annichini et. al provides abstract semi-algorithms for automated analysis of parametric counter and clock automata. However, they do not provide any general framework for deriving abstract semi-algorithms or analyzing their accuracy.

Colon and Uribe [CU98] present an algorithm that uses decision procedures to generate finite state abstractions of possibly infinite state systems. Our work is different from theirs; the denotation of the covering transformer monoid $Y = \langle \Phi', S' \rangle$ (i.e., $[\Phi']$) may be infinite; moreover $\Phi'$ may itself be infinite. In [CC98], Cousot and Cousot describe improvements to abstract model checking by combining forwards and backwards abstract fixpoint computations. It would be interesting to see how their techniques can be adapted to a constraint-based setting as ours. Cleaveland, Iyer and Yankelevich [CIY95] develop a framework in which they can establish optimality results by showing that a particular system abstraction is the most precise one possible among a class of safe abstractions. It is not clear how to apply their techniques in a constraint-based setting. An automata-theoretic framework for verification by finitary abstraction has been developed in [KPV99]. There, the authors reduce the verification problem to the infeasibility problem for Büchi discrete systems. They then provide a general proof method called WELL to establish the infeasibility of a Büchi discrete system. In contrast, our technique uses abstract symbolic forward analysis for verification after a covering has been established.

## References

[AAB00] A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems, 2000.

[ACHH93] R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems I*, LNCS 736, pages 209–229. Springer-Verlag, 1993.

[ACJT96] P. Abdulla, K. Cerans, B. Jonsson, and T. K. Tsay. General decidability theorems for infinite state systems. In *LICS*, pages 313–321, 1996.

[BBR97] B. Boigelot, L. Bronne, and S. Rassart. An improved reachability analysis method for strongly linear hybrid systems. In O. Grumberg, editor, *CAV'97: Computer Aided Verification*, volume 1254 of *LNCS*, pages 167–178. Springer-Verlag, 1997.

[BLL+96] Johan Bengtsson, Kim. G. Larsen, Fredrik Larsson, Paul Petersson, and Wang Yi. Uppaal in 1995. In T. Margaria and B. Steffen, editors, *TACAS*, LNCS 1055, pages 431–434. Springer-Verlag, 1996.

[Boi98] Bernard Boigelot. *Symbolic Methods for Exploring Infinite State Spaces*. PhD thesis, Universite De Liege, Montefiore, Belgium, 1998.

[CC77] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *the 4th ACM Symposium on Principles of Programming Languages*, 1977.

[CC98] P. Cousot and R. Cousot. Refining model checking by abstract interpretation. *Automated Software Engineering*, 6:69–95, 1998.

[CGJ+00] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In E. A. Emerson and A. P. Sistla, editors, *CAV: Computer-Aided Verification*, volume 1855 of *LNCS*, pages 154–169. Springer, 2000.

[CIY95] R. Cleaveland, P. Iyer, and D. Yankelevich. Optimality in abstractions of model checking. In A. Mycroft, editor, *SAS: Static Analysis Symposium*, volume 983 of *LNCS*, pages 51–63. Springer, 1995.

[CJ98] H. Comon and Y. Jurski. Multiple Counters Automata, Safety Analysis, and Presburger Arithmetics. In Alan J. Hu and M. Y. Vardi, editors, *CAV'98: Computer Aided Verification*, volume 1427 of *LNCS*, pages 268–279. Springer-Verlag, 1998.

[CU98] M. A. Colon and T. E. Uribe. Generating finite-state abstractions of reactive systems using decision procedures. In A. Hu and M. Y. Vardi, editors, *CAV: Computer-Aided Verification*, volume 1427 of *LNCS*, pages 293–304. Springer, 1998.

[DDP99] S. Das, D. L. Dill, and S. Park. Experience with predicate abstraction. In N. Halbwachs and D. Peled, editors, *CAV: Computer-Aided Verification*, volume 1633 of *LNCS*, pages 160–171. Springer, 1999.

[DP99] G. Delzanno and A. Podelski. Model Checking in CLP. In R. Cleaveland, editor, *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'99)*, volume 1579 of *LNCS*, pages 223–239. Springer-Verlag, March 1999.

[DT98] C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. In Bernhard Steffen, editor, *TACAS98: Tools and Algorithms for the Construction of Systems*, LNCS 1384, pages 313–329. Springer-Verlag, March/April 1998.

[Eil76] S. Eilenberg. *Automata, Languages and Machines*, volume B. Academic Press, 1976.

[GRS00] R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretations complete. *Journal of the Association of Computing Machinery (JACM)*, 47(2):361–413, March 2000.

[GS97] S. Graf and H. Saidi. Construction of abstract state graphs with pvs. In O. Grumberg, editor, *CAV: Computer-Aided Verification*, volume 1254 of *LNCS*, pages 72–83. Springer, 1997.

[HHWT97] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: a model checker for hybrid systems. In O. Grumberg, editor, *CAV97: Computer-aided Verification*, LNCS 1254, pages 460–463. Springer-Verlag, 1997.

[HKPV95] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *the 27th Annual Symposium on Theory of Computing*, pages 373–382. ACM Press, 1995.

[HPR97] N. Halbwachs, Y-E. Proy, and P. Romanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, 1997.

[KN00] R. P. Kurshan and K. S. Namjoshi. Syntactic program transformations for automatic abstractions. In E. A. Emerson and A. P. Sistla, editors, *CAV: Computer-Aided Verification*, volume 1855 of *LNCS*, pages 435–449. Springer, 2000.

[KPV99] Y. Kesten, A. Pnueli, and M. Y. Vardi. Verification by augmented abstraction: The automata-theoretic view. In J. Flum and M. R. Artalejo, editors, *CSL: Computer Science Logic*, volume 1683 of *LNCS*, pages 141–156. Springer, 1999.

[LPY95] K.G. Larsen, P. Pettersson, and W. Yi. Compositional and symbolic model checking of real-time systems. In *Proceedings of the 16th Annual Real-time Systems Symposium*, pages 76–87. IEEE Computer Society Press, 1995.

[LPY99] G. Lafferriere, G. J. Pappas, and S. Yovine. A new class of decidable hybrid systems. In F. W. Vaandrager and J. H. van Schuppen, editors, *Hybrid Systems, Computation and Control*, volume 1569 of *LNCS*, pages 137–151, 1999.

[MP99] S. Mukhopadhyay and A. Podelski. Beyond region graphs: Symbolic forward analysis of timed automata. In C. Pandurangan, V. Raman, and R. Ramanujam, editors, *19th International Conference on the Foundations of Software Technology and Theoretical Computer Science*, volume 1738 of *LNCS*, pages 233–245, December 1999.

[MP00a] S. Mukhopadhyay and A. Podelski. Accurate widenings and boundedness properties, 2000.

[MP00b] S. Mukhopadhyay and A. Podelski. Compositional termination analysis of symbolic forward analysis for infinite state systems, 2000.

[Sai00] H. Saidi. Model checking guided abstraction and analysis. In J. Palsberg, editor, *SAS: Static Analysis Symposium*, LNCS. Springer, 2000.

[SS99] H. Saidi and N. Shankar. Abstract and model check while you prove. In N. Halbwachs and D. Peled, editors, *CAV: Computer-Aided Verification*, volume 1633 of *LNCS*, pages 455–469. Springer, 1999.

## A Proofs

**Proof of Proposition 1** The equality follows directly from the definition of covering between constraint transformer monoids. Indeed, if $\varphi^2$ is reachable from $\varphi^1$, then there exists a $w \in \mathcal{E}^*$ such that $[\![w]\!](\varphi^1) = \varphi^2$. Since $f$ is a covering between $X$ and $Y$, there exists $v \in S'$ that covers $[\![w]\!]$. Hence, the equality follows from the definition.

Now assume the equality. If $f$ saturates $\varphi^1$ and one of the two conditions for homeocovering holds, then we show that there exists $w \in \mathcal{E}^*$ such that

$[\![w]\!](\varphi^1) = \varphi^2$. Suppose that the first condition holds. Since $f$ saturates $\varphi^1$, there must exist a constraint $\psi^1$ in $\Phi'$ such that $f(\varphi^1, \psi^1)$ and for all $\varphi'$ such that $f(\varphi', \psi^1)$, $\varphi^1 = \varphi'$, i.e., $\varphi^1 = \{\varphi' \,|\, f(\varphi', \psi^1)\}$. Also, by the assumed equality, $f(\varphi^2, v(\psi^1)$. Since $f^{-1}$ is a covering between $Y$ and $X'$, there exists a $w \in \mathcal{E}^*$ such that $[\![w]\!]$ covers $v$. Therefore $\{[\![w]\!](\varphi^1)\} = \{\varphi' \mid f(\varphi', v(\psi^1)\}$. Therefore $\varphi^2 = [\![w]\!](\varphi^1)$. Hence, $\varphi^2$ is reachable from $\varphi^1$.

Suppose now that the second condition holds. Let $X' = \langle \Phi'', S'' \rangle$ be an $f$-quotient of $X$ with $rep$ as the chosen representant function. Since $f$ saturates $\varphi^1$, there exists $\psi^1$ such that $f(\varphi^1, \psi^1)$ and for all $\varphi'$ such that $f(\varphi', \psi^1)$, we have $\varphi' = \varphi^1$. Since $\varphi^1$ and $\varphi^2$ are canonical with respect to $f$, we have $rep([\varphi^1]) = \varphi^1$ and $rep([\varphi^2]) = \varphi^2$. Hnce, we have, $f(rep([\varphi^1], \psi^1)$. By the assumed equality, there exists $v \in S'$ such that $f(\varphi^2, v(\psi^1))$. Since, $f^{-1}$ is a covering between $X$ and $X'$, there exists $\widetilde{[\![w]\!]} \in S''$, such that $\{\widetilde{[\![w]\!]}(rep([\varphi^1]))\} = \{rep([\varphi]) \mid f(rep([\varphi]), v(psi^1))\}$. Since, $rep([\varphi^2])$ is in the right hand side of this equality, therefore, $\widetilde{[\![w]\!]}(rep([\varphi^1])) = rep([\varphi^2])$. By canonicity of $\varphi^1$ and $\varphi^2$ with respect to $f$, $\varphi^2 = [\![w]\!](\varphi^1)$.

∎

**Proof of Theorem 1** The first statement follows from Proposition 1. Suppose that the inequality in the first statement of the theorem holds for all $i$. Seeking a contradiction, suppose that $\varphi^2$ is reachable from $\varphi^1$. Then, there exists $w_i \in S$ such that $\varphi^2 = w_i(\varphi^1)$. Now consider $\widetilde{w_i} \in S^{free}$ such that $w_i$ is the companion of $\widetilde{w_i}$. The node $\widetilde{w_i}$ in the abstract constraint tree $T_{\mathcal{C}}$ is labeled by $\Psi_i = \{v_i(\psi^0) \mid f(\varphi^1, \psi^0)\}$ where $v_i \in \mathcal{C}$ covers $w_i$. By Proposition 1, $\Psi_i = \{\psi \mid f(\varphi^2, \psi)\}$. Hence a contradiction.

Suppose, $f$ is a finitary covering with respect to $\varphi^1$. Consider, first, the case when $Y$ is finite. Then, along every branch of the abstract con the abstract constraint tree, there exists two nodes $\widetilde{w_i}$ and $\widetilde{w_j}$, where $j < i$, labeled by the same set of constraints $\Psi$. Hence, any abstract symbolic forward analysis terminates with local entailment. Suppose now that $Y$ is finitary with respect to $\{\psi^0 \mid f(\varphi^1, \psi^0)\}$. Then, along any branch of the constraint tree there exists a node $\widetilde{w_i}$ labeled by $\Psi_i$ such that for each constraint $\psi \in \Psi_i$, there exists a $j < i$ and a constraint $\psi' \in \Psi_j$ such that $\psi \models \psi'$. The statement 2 in the theorem follows from a direct application of Proposition 1.

∎

**Proof of Proposition 3** We first show that the constraint $trim(\varphi)$ cannot contain constraints of the form $x_i - x_j \sim a$ or $x_i \; relop \; a$ for all $i, j = 1, \ldots, n$, where $\sim \in \{>, \geq\}$, $relop \in \{>, <, \geq, \leq\}$, and $|a| > M$. Seeking a contradiction, suppose there exists a conjunct of the form $x_i - x_j > a$ where $|a| > M$. First suppose that $a > 0$. Then this conjunct is also present in $\varphi$. Suppose $\mathcal{R}, \boldsymbol{v} \models \varphi$. Then $v_i - v_j > a$. Therefore $v_i > a$. Therefore there exists no solution $\boldsymbol{v}$ of $\varphi$ such that $v_i \leq M$. So $\varphi \wedge x_i > M \equiv \varphi$. So the constraint is removed by the trim operation. Similarly for the case when $a < 0$. Now we write each constraint $x_i - x_j = c$ in the form $x_i - x_j \geq c \wedge x_i - x_j \leq c$. Similar for the case $x_i = c$. So given this representation, syntactically the number of distinct constraints is bounded by $(4M + 4)^{n(n-1)} \cdot (2M + 2)^{2n}$ which is $2^{O(n^2)} \cdot (2M + 2)^{O(n^2)}$.

This is because there are $n(n-1)$ pairs $x_i, x_j$. For each constraint of the form $x_i - x_j \sim c$, $\sim$ can be $>$ or $\geq$, and $c$ can take integral values from $-M$ to $M$. Also for each constraint $x_i \, relop \, c$, where $relop \in \{>, \geq\}$ (i.e., constraint determining the lower bound of a variable) $c$ can be a non-negative integer in the interval $[0, M]$. Similarly the case for the constraints determining the upper bound of a variable.

‖

**Proof of Proposition 6** Consider any $\varphi_i \in \Phi'$. Let $\psi = rep(\{\varphi \in \Phi \mid \varphi \models \varphi_i\})$ where $rep$ is a chosen representant function for a quotient. Consider $\widetilde{w} \in S'$. We claim that $[\![w]\!] \in S$ covers $\widetilde{w}$ with respect to $f^{-1}$. Suppose that $\widetilde{w}(\varphi_i) = \varphi_j$. It can be verified that $\widetilde{[\![w]\!]}(\psi) \models \varphi_j$. Therefore, in the $f$-quotient with the representant function $rep$, $\widetilde{[\![w]\!]}(\psi) = rep([[\![w]\!](\psi)]) = \psi'$. Therefore, by definition, $\psi' \models \varphi_j$. Therefore $f(\psi', \varphi_j)$. ‖

# Action Timing Discretization with Iterative-Refinement

Todd W. Neller*

Department of Computer Science
Gettysburg College
Gettysburg, PA 17325, USA
tneller@gettysburg.edu

**Abstract.** Artificial Intelligence search algorithms search discrete systems. To apply such algorithms to continuous systems, such systems must first be discretized, i.e. approximated as discrete systems. Action-based discretization requires that both action parameters and action timing be discretized. We focus on the problem of action timing discretization.
After describing an $\epsilon$-admissible variant of Korf's recursive best-first search ($\epsilon$-RBFS), we introduce iterative-refinement $\epsilon$-admissible recursive best-first search (IR $\epsilon$-RBFS) which offers significantly better performance for initial time delays between search states over several orders of magnitude. Lack of knowledge of a good time discretization is compensated for by knowledge of a suitable solution cost upper bound.

## 1 Introduction

Artificial Intelligence search algorithms search discrete systems, yet we live and reason in a continuous world. Continuous systems must first be discretized, i.e. approximated as discrete systems, to apply such algorithms. There are two common ways that continuous search problems are discretized: state-based discretization and action-based discretization. State-based discretization becomes infeasible when the state space is highly dimensional. Action-based discretization becomes infeasible when there are too many degrees of freedom. Interestingly, biological high-degree-of-freedom systems are often governed by a much smaller collection of motor primitives [3]. We focus here on action-based discretization.

Action-based discretization consists of two parts: (1) action parameter discretization and (2) action timing discretization, i.e. *how* and *when* to act. For example, consider robot soccer. Search can only sample action parameter continua such as kick force and angle. Similarly, search can only sample infinite possible action timings such as when to kick. The most popular form of discretization is uniform discretization. It is common to sample possible actions and action timings at fixed intervals.

In this paper, we focus on action timing discretization. Experimental evidence of this paper and previous studies [4] suggests that a fixed uniform discretization of time is not

* The author is grateful to Richard Korf for suggesting the sphere navigation problem, and to the anonymous AAAI and SARA reviewers for good insight and suggestions. This research was done both at the Stanford Knowledge Systems Laboratory with support by NASA Grant NAG2-1337, and at Gettysburg College.

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 170–177, 2002.

advisable for search if one has a desired solution cost upper bound. Rather, a new class of algorithms that dynamically adjust action timing discretization can yield significant performance improvements over static action timing discretization.

Iterative-refinement algorithms use a simple means of dynamically adjusting the time interval between search states. This paper presents the results of an empirical study of the performance of different search algorithms as one varies the initial time interval between search states. We formalize our generalization of search, present our chosen class of problems, describe the algorithms compared, and present the experimental results.

The key contributions of this work are experimental insight into the importance of searching with dynamic time discretization, and two new iterative-refinement algorithms, one of which exceeds the performance of $\epsilon$-RBFS across more than four orders of magnitude of the initial time delay between states.

## 2 Search Problem Generalization

Henceforth, we will assume that the action discretization, i.e. which action parameters are sampled, is already given. From the perspective of the search algorithm, the action discretization is static, i.e. cannot be varied by the algorithm. However, action timing discretization is dynamic, i.e. the search algorithm can vary the action timing discretization. For this reason, we will call such searches "SADAT searches" as they have Static Action and Dynamic Action Timing discretization.

We formalize the SADAT search problem as the quadruple:

$$\{S, s_0, A, G\}$$

where

- $S$ is the state space,
- $s_0 \in S$ is the initial state,
- $A = \{a_1, \ldots, a_n\}$ is a finite set of action functions $a_i : S \times \Re^+ \rightarrow S \times \Re$, mapping a state and a positive time duration to a successor state and a transition cost, and
- $G \subset S$ is the set of goal states.

The important difference between this and classical search formulations is the generalization of actions (i.e. operators). Rather than mapping a state to a new state and the associated cost of the action, we additionally take a time duration parameter specifying how much time passes between the state and its successor.

A goal path can be specified as a sequence of action-duration pairs that evolve the initial state to a goal state. The cost of a path is the sum of all transition costs. Given this generalization, the state space is generally infinite, and the optimal path is generally only approximable through a sampling of possible paths through the state space.

## 3 Sphere Navigation Problem

Since SADAT search algorithms will generally only be able to approximate optimal solutions, it is helpful to test them on problems with known optimal solutions. Richard

Korf proposed the problem of navigation between two points on the surface of a sphere as a simple benchmark with a known optimal solution.[1] Our version of the problem is given here.

The shortest path between two points on a sphere is along the great-circle path. Consider the circle formed by the intersection of a sphere and a plane through two points on the surface of the sphere and the center of the sphere. The *great-circle path* between the two points is the shorter part of this circle between the two points. The *great-circle distance* is the length of this path.

The state space $S$ is the set of all positions and headings on the surface of a unit sphere along with all nonnegative time durations for travel. Essentially, we encode path cost (i.e. time) in the state to facilitate definition of $G$. The initial state $s_0$ is arbitrarily chosen to have position (1,0,0) and velocity (0,1,0) in spherical coordinates, with no time elapsed initially.

The action $a_i \in A, 0 \leq i \leq 7$ takes a state and time duration, and returns a new state and the same time duration (i.e. cost = time). The new state is the result of changing the heading $i * \pi/4$ radians and traveling with unit velocity at that heading for the given time duration on the surface of the unit sphere. If the position reaches a goal state, the system stops evolving (and incurring cost).

The set of goal states $G$ includes all states that are both (1) within $\epsilon_d$ great-circle distance from a given position $p_g$, and (2) within $\epsilon_t$ time units of the optimal great-circle duration to reach such positions. Put differently, the first requirement defines the size and location of the destination, and the second requirement defines how directly the destination must be reached. Position $p_g$ is chosen at random from all possible positions on the unit sphere with all positions being equiprobable.

If $d$ is the great-circle distance between (1,0,0) and $p_g$, then the optimal time to reach a goal position at unit velocity is $d-\epsilon_d$. Then the solution cost upper bound is $d-\epsilon_d+\epsilon_t$. For any position, the great-circle distance between that position and $p_g$ minus $\epsilon_d$ is the optimal time to goal at unit velocity. This is used as the admissible heuristic function $h$ for all heuristic search.

## 4 Algorithms

In this section we describe the four algorithms used in our experiments. The first pair use fixed time intervals between states. The second pair dynamically refine time intervals between states. The first algorithm, $\epsilon$-admissible iterative-deepening A$^*$, features an improvement over the standard description. Following that we describe an $\epsilon$-admissible variant of recursive best-first search and two novel iterative-refinement algorithms.

### 4.1 $\epsilon$-Admissible Iterative-Deepening A*

$\epsilon$-admissible iterative-deepening A$^*$ search, here called $\epsilon$-IDA$^*$, is a version of IDA$^*$ [1] where the $f$-cost limit is increased "by a fixed amount $\epsilon$ on each iteration, so that the total number of iterations is proportional to $1/\epsilon$. This can reduce the search cost, at the expense of returning solutions that can be worse than optimal by at most $\epsilon$." [5].

[1] Personal communication, 23 May 2001.

Actually, our implementation is an improvement on $\epsilon$-IDA$^*$ as described above. If $\Delta f$ is the difference between (1) the minimum $f$-value of all nodes beyond the current search contour, and (2) the current $f$-cost limit, then the $f$-cost limit is increased by $\Delta f + \epsilon$. ($\Delta f$ is the increase that would occur in IDA$^*$.) This improvement is significant in cases where $f$-cost limit changes between iterations can significantly exceed $\epsilon$.

It is important to note that when we commit to an action timing discretization, the $\epsilon$-admissibility of search is relative to the optimal solution of this discretization rather than the optimal solution of the original continuous-time SADAT search problem.

### 4.2 $\epsilon$-Admissible Recursive Best-First Search

$\epsilon$-admissible recursive best-first search, here called $\epsilon$-RBFS, is an $\epsilon$-admissible variant of recursive best-first search that follows the description of [2, §7.3] without further search after a solution is found. As with our implementation of $\epsilon$-IDA$^*$, local search bounds increase by at least $\epsilon$ (when not limited by `B`) to reduce redundant search.

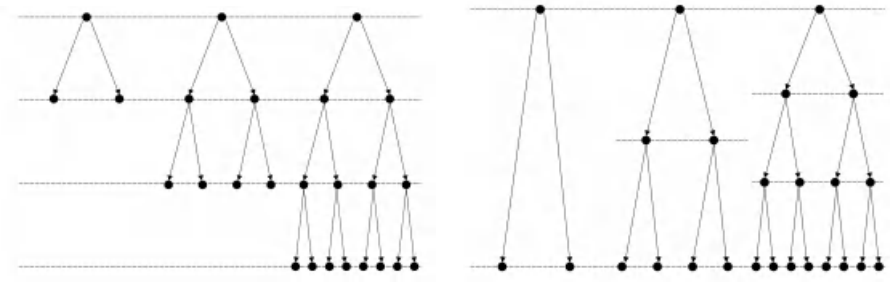
In Korf's style of pseudocode, $\epsilon$-RBFS is as follows:

```
eRBFS (node: N, value: F(N), bound: B)
IF f(N)>B, RETURN f(N)
IF N is a goal, EXIT algorithm
IF N has no children, RETURN infinity
FOR each child Ni of N,
  IF f(N)<F(N), F[i] := MAX(F(N),f(Ni))
  ELSE F[i] := f(Ni)
sort Ni and F[i] in increasing order of F[i]
IF only one child, F[2] := infinity
WHILE (F[1] <= B and F[1] < infinity)
  F[1] := eRBFS(N1, F[1], MIN(B, F[2] + epsilon))
  insert Ni and F[1] in sorted order
RETURN F[1]
```

The difference between RBFS and $\epsilon$-RBFS is in the computation of the bound for the recursive call. In RBFS, this is computed as `MIN(B, F[2])` whereas in $\epsilon$-RBFS, this is computed as `MIN(B, F[2] + epsilon)`. `F[1]` and `F[2]` are the lowest and second-lowest stored costs of the children, respectively. A correctness proof of $\epsilon$-RBFS follows the structure of Korf's RBFS correctness proof [2] with minor modifications.

The algorithm's initial call parameters are the root node $r$, $f(r)$, and $\infty$. Actually, both RBFS and $\epsilon$-RBFS can be given a finite bound $b$ if one wishes to restrict search for solutions with a cost of no greater than $b$ and uses an admissible heuristic function. If no solution is found, the algorithm will return the $f$-value of the minimum open search node beyond the search contour of $b$.

In the context of SADAT search problems, both $\epsilon$-IDA$^*$ and $\epsilon$-RBFS assume a fixed time interval between a node and its child. The following two algorithms do not.



(a) Iterative-deepening DFS. (b) Iterative-refinement DFS.

**Fig. 1.** Iterative search methods.

### 4.3 Iterative-Refinement $\epsilon$-RBFS

Iterative-refinement [4] is perhaps best described in comparison to iterative-deepening. Iterative-deepening depth-first search (Figure 1(a)) provides both the linear memory complexity benefit of depth-first search and the minimum-length solution-path benefit of breadth-first search at the cost of node re-expansion. Such re-expansion costs are generally dominated by the cost of the final iteration because of the exponential nature of search time complexity.

Iterative-refinement depth-first search (Figure 1(b)) can be likened to an iterative-deepening search to a fixed time-horizon. In classical search problems, time is not an issue. Actions lead from states to other states. When we generalize such problems to include time, we then have the choice of how much time passes between search states. Assuming that the vertical time interval in Figure 1(b) is $\Delta t$, we perform successive searches with delays $\Delta t$, $\Delta t/2$, $\Delta t/3$, ... until a goal path is found.

Iterative-deepening addresses our lack of knowledge concerning the proper depth of search. Similarly, iterative-refinement addresses our lack of knowledge concerning the proper time discretization of search. Iterative-deepening performs successive searches that grow exponentially in time complexity. The complexity of previous unsuccessful iterations is generally dominated by that of the final successful iteration. The same is true for iterative-refinement.

However, the concept of iterative-refinement is not limited to the use of depth-first search. Other algorithms such as $\epsilon$-RBFS may be used as well. In general, for each iteration of an iterative-refinement search, a level of (perhaps adaptive) time-discretization granularity is chosen for search and an upper bound on the solution cost is given. If the iteration finds a solution within this cost bound, the algorithm terminates with success. Otherwise, a finer level of time-discretization granularity is chosen, and search is repeated. Search is successively refined with respect to time granularity until a solution is found.

Iterative-Refinement $\epsilon$-RBFS is one instance of such search. The algorithm can be simply described as follows:

```
IReRBFS (node: N, bound: B, initDelay: DT)
FOR I = 1 to infinity
  Fix the time delay between states at DT/I
  eRBFS(N, f(N), B)
  IF eRBFS exited with success, EXIT algorithm
```

Iterative-Refinement $\epsilon$-RBFS does not search to a fixed time-horizon. Rather, each iteration searches within a search contour bounded by $B$. Successive iterations search to the same bound, but with finer temporal detail. `DT/I` is assigned to a global variable governing the time interval between successive states in search.

### 4.4 Iterative-Refinement DFS

The algorithm for Iterative-Refinement DFS is given as follows:

```
IRDFS (node: N, bound: B, initDelay: DT)
FOR I = 1 to infinity
  Fix the time delay between states at DT/I
  DFS-NOUB(N, f(N), B)
  IF DFS-NOUB exited with success, EXIT algorithm
```

Our depth-first search implementation DFS-NOUB uses a node ordering (NO) heuristic and has a path cost upper-bound (UB). The node-ordering heuristic is as usual: Nodes are expanded in increasing order of $f$-value. Nodes are not expanded that exceed a given cost upper bound. Assuming admissibility of the heuristic function $h$, no solutions within the cost upper-bound will be pruned from search.

## 5 Experimental Results

In these experiments, we vary only the initial time delay $\Delta t$ between search states and observe the performance of the algorithms we have described. For $\epsilon$-IDA$^*$ and $\epsilon$-RBFS, the initial $\Delta t$ is the only $\Delta t$ for search. The iterative-refinement algorithms search using the harmonic refinement sequence $\Delta t$, $\Delta t/2$, $\Delta t/3$, . . ., and are limited to 1000 refinement iterations. $\epsilon$-admissible searches were performed with $\epsilon = .1$.

Experimental results for success rates of search are summarized in Figure 2. Each point represents 500 trials over a fixed, random set of sphere navigation problems with $\epsilon_d = .0001$ and $\epsilon_t$ computed as 10% of the optimal time. Thus, the target size for each problem is the same, but the varying requirement for solution quality means that different delays will be appropriate for different search problems. Search was terminated after 10 seconds, so the success rate is the fraction of time a solution was found within the allotted time and refinement iterations.

In this empirical study, means and 90% confidence intervals for the means were computed with 10000 bootstrap resamples.

Let us first compare the performance of iterative-refinement (IR) $\epsilon$-RBFS and $\epsilon$-RBFS. To the left of the graph, where the initial $\Delta t_0$ is small, the two algorithms have identical behavior. This region of the graph indicates conditions under which a solution is

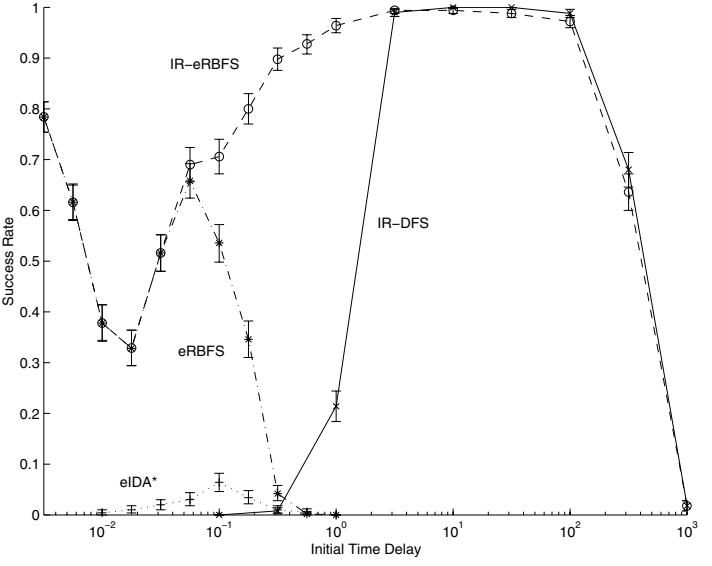



**Fig. 2.** Effect of varying initial $\Delta t$.

found within 10 seconds on the first iteration or not at all. There is no iterative- refinement in this region; the time complexity of the first iteration leaves no time for another.

At about $\Delta t_0 = .1$, we observe that IR $\epsilon$-RBFS begins to have a significantly greater success rate than $\epsilon$-RBFS. At this point, the time complexity of search allows for multiple iterations, and thus we begin to see the benefits of iterative-refinement.

Continuing to the right with greater initial $\Delta t_0$, IR $\epsilon$-RBFS nears a 100% success rate. At this point, the distribution of $\Delta t$'s over different iterations allows IR $\epsilon$-RBFS to reliably find a solution within the time constraints. We can see the distribution of $\Delta t$'s that most likely yield solutions from the behavior of $\epsilon$-RBFS.

Where the success rate of IR $\epsilon$-RBFS begins to fall, the distribution of first 1000 $\Delta t$'s begins to fall outside of the region where solutions can be found. With our refinement limit of 1000, the last iteration uses a minimal $\Delta t = \Delta t_0/1000$. The highest $\Delta t_0$ trials fail not because time runs out. Rather, the iteration limit is reached. However, even with a greater refinement limit, we would eventually reach a $\Delta t_0$ where the iterative search cost incurred on the way to the good $\Delta t$ range would exceed 10 seconds.

Comparing IR $\epsilon$-RBFS with IR DFS, we first note that there is little difference between the two for large $\Delta t_0$. For mid-to-low-range $\Delta t_0$ values, however, we begin to see the efficiency of $\epsilon$-RBFS over DFS with node ordering as the first iteration with a solution path presents a more computationally costly search. Without a perfect heuristic where complex search is necessary, $\epsilon$-RBFS shows its strength relative to DFS. Rarely will problems be so unconstrained and offer such an easy heuristic as this benchmark

problem, so IR $\epsilon$-RBFS will be generally be better suited for all but the simplest search problems.

In summary, iterative-refinement algorithms are statistically the same as or superior to the other searches over the range of $\Delta t_0$ values tested. IR $\epsilon$-RBFS offers the greatest average success rate across all $\Delta t_0$. With respect to $\epsilon$-RBFS, IR $\epsilon$-RBFS offers significantly better performance for $\Delta t_0$ spanning more than four orders of magnitude. These findings are in agreement with previous empirical studies concerning a submarine detection avoidance problem [4].

## 6 Conclusions

This empirical study concerning sphere navigation provides insight into the importance of searching with dynamic time discretization. Iterative-refinement algorithms are given an initial time delay $\Delta t_0$ between search states and a solution cost upper bound. Such algorithms iteratively search to this bound with successively smaller $\Delta t$ until a solution is found.

Iterative-refinement $\epsilon$-admissible recursive best-first search (IR $\epsilon$-RBFS) was shown to be similar to or superior to all other searches studied for $\Delta t_0$ spanning over five orders of magnitude. With respect to $\epsilon$-RBFS (without iterative-refinement), a new $\epsilon$-admissible variant of Korf's recursive best-first search, IR $\epsilon$-RBFS offers significantly better performance for $\Delta t_0$ spanning over four orders of magnitude.

Iterative-refinement algorithms are important for search problems where reasonable values for $\Delta t$ are (1) unknown or (2) known and one wishes to find a solution more quickly and reliably. The key tradeoff is that of knowledge. Lack of knowledge of a good time discretization is compensated for by knowledge of a suitable solution cost upper bound. If one knows a suitable solution cost upper bound for a problem where continuous time is relevant, an iterative-refinement algorithm such as IR $\epsilon$-RBFS is recommended.

## References

1. Richard E. Korf. Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.
2. Richard E. Korf. Linear-space best-first search. *Artificial Intelligence*, 62:41–78, 1993.
3. Maja J. Mataric. Sensory-motor primitives as a basis for imitation: Linking perception to action and biology to robotics. In C. Nehaniv and K. Dautenhahn, editors, *Imitation in Animals and Artifacts*, Cambridge, MA, USA, 2000. MIT Press. See also USC technical report IRIS-99-377.
4. Todd W. Neller. *Simulation-Based Search for Hybrid System Control and Analysis*. PhD thesis, Stanford University, Palo Alto, California, USA, June 2000. Available as Stanford Knowledge Systems Laboratory technical report KSL-00-15 at www.ksl.stanford.edu.
5. Stuart Russell and Peter Norvig. *Artificial Intelligence: a modern approach*. Prentice Hall, Upper Saddle River, NJ, USA, 1995.

# Formalizing Approximate Objects and Theories: Some Initial Results

Aarati Parmar

Stanford University, Stanford, CA 94305, USA
aarati@cs.stanford.edu,
http://www-formal.stanford.edu/aarati

**Abstract.** This paper introduces some preliminary formalizations of the approximate entities of [McCarthy, 2000]. Approximate objects, predicates, and theories are considered necessary for human-level AI, and we believe they enable very powerful modes of reasoning (which admittedly are not always sound). Approximation is known as *vagueness* in philosophical circles and is often deplored as a defective aspect of human language which infects the precision of logic. Quite to the contrary, we believe we can tame this monster by formalizing it within logic, and then can "build solid intellectual structures on such swampy conceptual foundations." [McCarthy, 2000].
We first introduce various kinds of approximation, with motivating examples. Then we develop a simple ontology, with minimal philosophical assumptions, in which to cast our formalization. We present our formalization, and show how it captures some ideas of approximation.

## 1 Introduction

[McCarthy, 2000] introduces approximate objects, predicates, and theories as an extension to AI. Informally, an entity is *approximate* if it can be further refined by finding out more things about it, or by simply defining more. Reasoning with approximate entities ignores unnecessary details, thereby simplifying and accelerating reasoning in general, while remaining somewhat sound. Common sense reasoning will require exactly this property. As in the Aristotle quote of [McCarthy, 2000], "Our discussion will be adequate if it has as much clearness as the subject matter admits of; for precision is not to be sought for alike in all discussions".

We also need to formalize approximation so that we know its boundaries; one needs to know when an approximation fails, and how to move to the next level of precision to reason correctly again. As an example, consider any of the common sense problems displayed in [Morgenstern, 1998], such as cracking an egg. Any theory explaining this process will be inherently approximate, as formalizing every eventuality is tedious and maybe impossible. In general we always feel uncomfortable with any formalization, pointing out its inapplicability with

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 178–195, 2002.

respect to one eventuality or another.[1] The point is, if we can understand how a theory is approximate, then we can accept its failures rather than decry its deficiencies, and simply move on to the next more precise theory when necessary.

Incidentally, many common sense theories of the world use the abnormality predicate $Ab$ and second-order minimization to be robust outside the boundaries of approximation. For example, $\neg Ab(x) \implies (Bird(x) \implies Flies(x))$ formalizes "All birds fly," which, according to the $Ab$-minimization will infer $Bird(x) \implies Flies(x)$ when consistent to do so. Otherwise it will infer $Ab(x)$, as in the case of a penguin. Other than in action theories and inheritance hierarchies, one never questions how to insert the $Ab$s in a common sense theory, or how to refine them when faced with new information. We hope to provide some preliminary results that will help answer these two questions.

We believe our results may be applicable to other common-sense theories which formalize aspects of the world. Consider any simple grammar describing formation of English sentences ($S \leftarrow NP\ VP, \ldots$). It will correctly discriminate a restricted class of sentences, but will have to be increasingly complex as it approaches the full generality of English. Similar phenomena occur for other linguistic theories. However humans, when asked to explain their machinations, give a very simple illustration, adding elaborations only when necessary. This suggests that the formalization of linguistic concepts is not some grand monolithic (and highly complicated) theory but perhaps a series of approximate theories, each expanding and elaborating upon previous theories. This structure would be a lot more elaboration tolerant [McCarthy, 1998] as well.

A contrasting view [Dreyfus and Dreyfus, 1984] asserts that humans do not use rules, but rather discriminate "thousands of special cases." This work describes skill acquisition, the process by which a human masters a domain, as first using a simple set of rules, but ending up with a discriminator of many subtle cases based on experience. The expert does not consciously know these discriminations, and when asked to give rules explaining his behavior, will revert to the basic rules learned as a novice. This explains why an expert system, whose rules are acquired through interviews with experts, are competent but do not perform as well as the experts whose rules it is using. If [Dreyfus and Dreyfus, 1984] describes the true model of human skill acquisition, this would also explain why discriminatory structures such as decision trees, and neural networks have been so popular in AI, as they are models of real-world processes. If this is the case, then at least our theory will formally explain how simple rules get elaborated to a complex structure such as a neural network. If we can interpret the resulting structure as compiled rules, then perhaps our formalism could show how to extract out the declarative versions of these compiled rules. These intriguing avenues are beyond the scope of this preliminary paper.

[1] This is probably the reason why much of AI has focused on toy examples. The examples approximate the world in such a way that minimizes our guilt about how simple the theory is.

### 1.1 Philosophers' Views on Approximation

We mentioned that what we call approximation, many philosophers in the literature denote as *vagueness*. In general a concept is vague if it has borderline cases, and the boundaries of the indeterminacy are themselves blurred, as are the boundaries of those, and so on [Sorensen, 1997]. Examples given in the literature include baldness, the physical extent of Mount Everest, and the revival of a club (due to Parfit, in [Broome, 1984]). The philosophers debate whether vagueness exists in the world, or is just a linguistic artifact. Any discussion of vagueness involves whether a sharp boundary line truly exists for vague concepts, illustrated by the *sorites paradox*, that set of arguments which states:

1. 1 grain does not make a heap.
2. If $n$ grains do not make a heap, then neither does $n + 1$ grains.
3. $\therefore$ $10^6$ grains do not make a heap.

Since this argument is false, there must be a point at which $n$ grains are not a heap, but adding 1 more does, which entails that some sharp boundary exists between being a heap and not being a heap.

One of the proposed formalizations of vagueness is *fuzzy logic*, which instead of ascribing true or false values to a sentence, ascribes a value between 0 and 1. Hence if $x$ is only somewhat bald (a borderline case) we might ascribe to the sentence "$x$ is bald" the value 0.5. This value is referred to as the *degree of truth* of the sentence. It seems however, that assigning a vague sentence an absolute value is a bit paradoxical; fuzzy logic avoids this by also allowing *higher types*. That is, instead of assigning a value, one can assign an interval of possible values. Any interval $[a, b]$ can transformed into a fuzzier interval $[[a_1, a_2], [b_1, b_2]]$, where both boundaries of the original interval are fuzzified into intervals themselves. And these can be fuzzified further. If one does not like numbers, one can use any lattice of elements. It turns out that the actual values of numbers are not very important in practice anyway [Goguen, 1968].

Another proposal asserts that vagueness is linguistic, in that some concepts (like baldness) are simply vaguely specified and can have various underlying precise definitions. For example one could define baldness in various ways, as having 0 hairs on one's scalp, having at most 10 hairs, etc. Each one of these possible definitions is a *sharpening* of the concept of baldness, and described with a three-valued logic: a sentence containing a vague concept is true if it holds under all sharpenings of the concept, false if it is false under all sharpenings of the concept, and "indefinite" if it is true for only some of the sharpenings. But then, unfortunately, the law of excluded middle ($\phi \vee \neg\phi$) won't hold for borderline sentences $\phi$.[2]

A third view notices that in the previous case, a meta-language is used to give definite truth-conditions for the vaguer target language (although they themselves may be indefinite), and therefore definitely shows at what point truth and falsity dissolve into indeterminacy. So for example in the sorites paradox, at

[2] The argument may be found in [Tye, 2000].

some point whether $n$ grains of sand make a heap will be a truth, and at $n-1$ it will be indeterminate. The fact that there is a sharp line where indeterminacy starts opposes the definition of vagueness. [Tye, 2000] proposes that even the meta-language will need to be vague, as well as the meta-meta-language, etc. This regress is known as higher-order vagueness. (Note that fuzzy logic could model this regress using higher types.)

Finally the *epistemic* view [Williamson, 1994] espouses that definite boundaries for vague concepts exist, but we just cannot know them. Vagueness is an ignorance which we can never overcome.

### 1.2 Related Work on Abstraction

[Giunchiglia and Walsh, 1992] is one of the first attempts to formalize *abstraction*, defined as the process of mapping the representation of a problem to another one, so that the problem is easier to solve. It also shows how various paradigms, such as ABSTRIPS, [Hobbs, 1985]'s theory of granularity, etc. are instances of various kinds of abstractions. We share the same motivations as [Giunchiglia and Walsh, 1992] in that we both care about the class of mappings that preserve desirable properties while throwing away unnecessary details. However we differ in approach in that we do not concentrate on syntactic mappings between two axiomatic formal systems, which preserve the set of (non-)theorems in some way. While this is important when we discuss *approximate theories*, we care more about the fine structure of such theories, which requires an examination of the nature of objects and predicates in the theory. We want to know what are the intrinsic qualities about theories that make them so amenable to certain forms of abstraction.

### 1.3 Propositional Approximate Theories

We repeat the treatment of propositional approximate theories given in [McCarthy, 2000]. The ontology includes reality, modeled by a set of propositional variables $r_1, \ldots, r_n$. There are so many observations that can be made, denoted by $o_1, \ldots, o_k$, which are each propositional functions of reality:

$$o_i = \mathcal{O}_i(r_1, \ldots, r_n). \tag{1}$$

The functions model the fact that much of reality is not directly observable. The fact that $n \gg k$ reflects the complexity of reality versus our observations.

$q_1, \ldots, q_l$ are a set of propositions about reality whose values we are interested in learning. They are determined by reality:

$$q_i = \mathcal{Q}_i(r_1, \ldots, r_n). \tag{2}$$

Our theory $\mathcal{AT}$ only approximates $\mathcal{Q}_i(r_1, \ldots, r_n)$ with $\mathcal{Q}'_i(o_1, \ldots, o_k)$, which are only functions of our indirect observations about reality, and not reality itself like the $\mathcal{Q}_i$s.

$Lucky(r_1, \ldots, r_n)$ is a predicate which when true, implies that our approximations to the $q_i$s is correct:

$$Lucky(r_1, \ldots, r_n) \implies (\forall i \leq l)[\mathcal{Q}_i(r_1, \ldots, r_n) = \mathcal{Q}'_i(\mathcal{O}_1(r_1, \ldots, r_n), \ldots, \mathcal{O}_k(r_1, \ldots, r_n))]. \tag{3}$$

The main points of this formalization is that:

1. Reality is not always directly observable, partly because of its complexity, and partly because of the sheer enormity of details involved. The idea is that facts about reality may not be *epistemologically adequate*,[3] and the observations $o_i$ are used to express what the subject can in fact sense.
2. The difference in cardinality between $n$, and $k$ and $l$ also reflects this notion that reality is rich, and our observations are poor. Information is generally lost in the transition from reality to observations. This cardinality of facts is only one way to formalize this idea, and definitely not the most general.
3. Unfortunately, while the $q_i$ are functions of reality, we can only cobble together approximate functions based on our observations. Since our observations are usually lossy compared to reality, it is unlikely that the $\mathcal{Q}_i$ and $\mathcal{Q}'_i$ will coincide. (Only if we are *Lucky*!)

### 1.4 The Ontology of Approximate Things

Here we further interpret the approximate objects, predicates, and theories introduced in [McCarthy, 2000]. An *approximate object* is an object $o$ in a logical theory $T$, which either only partially captures the properties of some real object in *reality* or is itself inherently partial. The approximateness is inherent to the object, and not a matter of incompleteness of the theory (although to maintain consistency a theory with approximate objects may have to abandon completeness). Rather it is more a fact that the theory or its language may not be able to properly capture an object's properties. We can attempt a definition of approximate objects by first expounding three kinds of approximations of objects that occur in common sense reasoning about the world:

1. (TYPE I) The first kind of approximate object is that which represents an epistemologically richer[4] object. For example, most Blocksworld theories idealize a block so extremely as to ignore its physical characteristics, as well as other relevant properties, such as its mass. Such an approximation generally ignores irrelevant properties as well, such as the color of the block, or where it was manufactured. It is clear that this sort of abstraction is useful

[3] "[McCarthy and Hayes, 1969] defines an epistemologically adequate representation of information as one that can express the information actually available to a subject under given circumstances." from [McCarthy, 1979]

[4] [Howe, 1994] describes *rich* objects as those for which can be asserted properties, which we cannot be completely described. *Poor* objects are exactly the opposite.

in cases, to strain out many [irrelevant] facts, to simplify the ontology to the task at hand. Approximations are most useful when an epistemologically rich object is approximated by a poor one, to eliminate an entire order of complexity.

2. (TYPE II) Another kind of approximation is the mental conception of objects for which there is no basis in reality. One example includes the concept of being middle-aged. This is an abstraction constructed by humans, which has no corresponding concept in reality: there are facts about being "middle-aged" for which there is no inherent truth, and can only be decided by *defining more*.
   Another example is the concept of being red. In reality, there is no such concept; there are only wavelengths, generally between 600 and 700 nanometers, that give us humans the sensation of seeing red. Baldness, and what a heap is, are other such approximations, and the paradoxes that arise from studying them too closely arise from the fact that there is no real corresponding concept in reality that would decide the matter.
   This kind of approximation can be thought of as a human-created characterization of some phenomenon, which either simplifies reasoning or organizes it nicely. These kinds of approximation can be used to characterize a wide range of categories, without wasting too many words on a complete specification. Because they are not completely specified, the definitions will be incoherent with respect to reality, often leading to paradoxes like that of the heap. Since the definitions are defined by humans they are subject to cultural as well as personal biases.
3. (TYPE III) The final type of approximate objects are those that arise in counterfactual sentences [Costello and McCarthy, 1999]. Like Type II objects, they have no direct analogy in reality, by definition. They are only defined by whatever properties that are ascribed to them in the counterfactual. There is no truth of the matter for any other properties not mentioned in (or derivable from) the counterfactual.
   For example in "If another car had come over the hill when you passed that car, there would have been a head-on collision," the other car could have been a Buick, a Mercedes, etc. There is no truth to the matter about what make of car it was.

An *approximate predicate* is one whose extent is vague or ill-defined. There are borderline cases whose membership is questionable, and therefore it is difficult to come up with necessary and sufficient conditions. Some illustrative examples include "the wants of the U.S." [McCarthy, 2000] and religion [Alston, 1967]. We define *concepts* as unary predicates, which can be approximate, such as natural kinds (is an orange lemon still a lemon?). Fuzzy logic has had success in this area, as it allows one to talk about the *degree of membership* of an object in a set, which can represent borderline cases.

We hope to address the relation between our formalism and fuzzy logic in later work. Our intuition is that a logically defined concept $\phi$ will be vague because it is fundamentally ill-defined or incoherent, such as baldness or heaps.

Fuzzy logic can handle the questionable cases by ascribing partial degrees of truth. In first order logic, the only recourse is to either have an incomplete theory, in the formal sense that for some object $c$, $T \not\models bald(c) \wedge T \not\models \neg bald(c)$, or to restrict the theory to some limited domain of discourse. (such as the set of people who are either definitely bald or not.) A much better approach would explicate how and when a concept is ill-defined, with respect to a more accurate theory. This would then be superior to the fuzzy logic approach, since it would explain *why* borderline cases are borderline, rather than sweeping the problem under the rug by ascribing a partial degree of truth.

An *approximate theory* is a set of sentences which is an abbreviated description (in some sense) of some phenomenon. One example is when unnecessary details about the world are ignored: reasoning about what will be served for lunch on a flight is not required to plan a trip to Hawaii. Another is a kind of idealization such as Blocksworld. We give a framework in which to talk about theory approximation in §3.

The rest of this paper follows this outline: after some mathematical notation is introduced (§2), we delve into a proposed ontology for first order theories (§3), and then a formal definition of when one theory approximates another (§4), with some examples. §5 formalizes epistemologically rich and poor objects, which enables us to address the different types of approximate objects (§6). Finally in §7 we consider when an approximate theory is *coherent* (correctly predicts reality) and conclude in §8.

## 2 Preliminaries

We describe our notation and simplifications here. $\mathfrak{A}$, $\mathfrak{B}$, $\mathfrak{R}$, $\mathfrak{M}$, and $\mathfrak{M}'$ are first-order structures with non-empty universes. The universe of $\mathfrak{A}$ is denoted $|\mathfrak{A}|$. $\mathbf{M}$ denotes a class of first-order structures. $T, T_A, T_B$ are all consistent first-order theories. Any symbol of the form $\mathcal{L}_{\mathcal{X}}$ is a signature, or language.

We often interchange, in proofs and examples, a class of models $\mathbf{M}$ with the theory $T$ describing them. This should not be a source of confusion, as we know there are well-defined functions $Th_{\mathcal{L}}(\mathbf{M})$ which given a class of models in language $\mathcal{L}$ returns the set of formulas true in all of them, and $Mod(T)$ which returns the class of models of $T$.

Finally, if $X$ is a set, then $X^n$ is the set of $n$-tuples of $X$. $\mathfrak{A} \cong \mathfrak{B}$ means that there is an isomorphism between the structures $\mathfrak{A}$ and $\mathfrak{B}$. If two models are isomorphic then they entail the same set of sentences.

## 3 Our Ontology

In this section we would like to construct a formalism which reflects the intuitions in §1.3, but is more general, and allows first-order statements rather than propositions. To this end, we introduce the following ontology:

Let $\mathfrak{R}$ be a first-order model of (one's idea of) reality, using some language $\mathcal{L}_{\mathcal{R}}$. We let $\mathbf{R}$ be the collection or class of such models. On the other hand, our

observations are constructed in the language $\mathcal{L}_{\mathcal{O}}$. The language $\mathcal{L}_{\mathcal{O}}$ is determined by the observations that can be made by our sensory organs. The standard set of philosophical inquiries that we could make about the existence of the models of reality $\mathbf{R}$ is moot, since each element $\mathfrak{R} \in \mathbf{R}$ is supposed to be the observer's *perception* or idealization of what reality is, which may be isomorphic to reality, but not necessarily so. So for example, to a particle physicist, $\mathfrak{R}$ would include interactions between quarks. The $\mathfrak{R}$ of an ancient Greek would explain weather patterns in terms of Zeus' temper and throwing lightning bolts.

While $\mathcal{L}_{\mathcal{O}}$ must be *epistemologically adequate*, $\mathcal{L}_{\mathcal{R}}$ should be *metaphysically adequate*.[5] We use a class of models instead of a sole model to represent different perceptions of reality (wave versus particle interpretation of light), or incompleteness in one's perception of reality. If there is no such incompleteness, then $\mathbf{R}$ can just be $\{\mathfrak{R}\}$, the singleton model.

The relation between $\mathcal{L}_{\mathcal{O}}$ and $\mathcal{L}_{\mathcal{R}}$ not only explains how objects are mapped from one's sensations to reality (in other words, how they are grounded), but impart structure to any theory of reality based on our observations. $\mathcal{L}_{\mathcal{R}}$ describes very rich phenomena, which explains why $\mathcal{L}_{\mathcal{O}}$, which is meant to be a relatively simple language, does not coincide with it. Note that instead of providing an axiomatization of $\mathbf{R}$, we provide the models themselves. This is because there may not be a finite first-order axiomatization of $\mathbf{R}$. We also prefer classes of models to a theory, because assaying truth in reality appears to be more of a determination of whether it holds in the world (whether $\mathfrak{R} \models \phi$), rather than some process of inference based on statements (whether $\mathfrak{R} \vdash \phi$).

## 4 Theories Approximating Theories

The method of *syntactic interpretation* used by [Tarski et al., 1953] used to prove [un]decidability of theories can be used to express whether one theory approximates another. We copy the presentation in [Baudisch et al., 1985]: given two classes of models $\mathbf{A}$ and $\mathbf{B}$, respectively, of languages $\mathcal{L}_{\mathcal{A}}$ and $\mathcal{L}_{\mathcal{B}}$, we define an interpretation $I : \mathcal{L}_{\mathcal{A}} \to \mathcal{L}_{\mathcal{B}}$ which sends every predicate symbol $P(\overline{x})$ of $\mathcal{L}_{\mathcal{A}}$ to the formula $\phi_P(\overline{x})$ in $\mathcal{L}_{\mathcal{B}}$, and the formula $x = x$ to $\phi_=(x)$, which is a formula of $\mathcal{L}_{\mathcal{B}}$. For now we assume that we only have relational symbols, and no functions or constants in $\mathcal{L}_{\mathcal{A}}$, as they can be represented in terms of relations and some extra axioms. We explain the transformation for constant and function symbols in §4.1.

This interpretation is extended to all formulas in $\mathcal{L}_{\mathcal{A}}$ using the following inductive rules, where $\alpha$ and $\beta$ are formulas of $\mathcal{L}_{\mathcal{A}}$, $x$ and $y$ variables, and $\overline{x}$ a tuple of variables:

1. $(x = y)^I = x = y$
2. $(P(\overline{x}))^I = \phi_P(\overline{x})$

[5] [McCarthy, 1979] defines metaphysically adequate representations as those "that can represent complete facts ignoring the subject's ability to acquire the facts in given circumstances."

3. $(\neg\alpha)^I = \neg(\alpha)^I$
4. $(\alpha \vee \beta)^I = \alpha^I \vee \beta^I$
5. $(\exists x\alpha)^I = (\exists x)(\phi_=(x) \wedge \alpha^I)$

From any model $\mathfrak{B} \in \mathbf{B}$ in the target language $\mathcal{L}_\mathcal{B}$, we can define a structure $\mathfrak{B}^I$ in $\mathcal{L}_\mathcal{A}$. This consists of two simple steps:

1. We simply set the universe of $\mathfrak{B}^I$ to be the set of elements in $\mathfrak{B}$ obeying $\phi_=(x)$:

$$|\mathfrak{B}^I| = \{x \in |\mathfrak{B}| \mid \mathfrak{B} \models \phi_=(x)\}. \tag{4}$$

2. Then we map the interpretation of each $n$-ary predicate $P$ of $\mathcal{L}_\mathcal{A}$:

$$P^{\mathfrak{B}^I} = \{\overline{x} \in |\mathfrak{B}^I|^n \mid \mathfrak{B} \models \phi_P(\overline{x})\}. \tag{5}$$

For each predicate $P \in \mathcal{L}_\mathcal{A}$, $\mathfrak{B}^I$ "back-translates" a possible definition for it by looking at $\phi_P$ in $\mathfrak{B}$. $P$'s domain of application is controlled by the predicate $\phi_=$, which picks out the part of $\mathfrak{B}$ corresponding to objects in $T_A$.

One can finally define a notion of *interpretability*:

**Definition 1 (Interpretability).** *Let $T_A = Th_{\mathcal{L}_\mathcal{A}}(\mathbf{A})$ and $T_B = Th_{\mathcal{L}_\mathcal{B}}(\mathbf{B})$. Then the theory $T_A$ is* interpretable *in $T_B$, or $T_B$* interprets *$T_A$ if there is an interpretation function $I : \mathcal{L}_\mathcal{A} \to \mathcal{L}_\mathcal{B}$ such that:*

1. *for every structure $\mathfrak{B} \in \mathbf{B}$ there is a $\mathfrak{A} \in \mathbf{A}$ such that $\mathfrak{B}^I \cong \mathfrak{A}$ and,*
2. *for every structure $\mathfrak{A} \in \mathbf{A}$ there is a $\mathfrak{B} \in \mathbf{B}$ such that $\mathfrak{B}^I \cong \mathfrak{A}$.*

The two requirements for interpretability can be explained as (1): every model of $T_B$ can be "back-translated" by $I$ to be a model of $T_A$, and (2): every model of $T_A$ can be expanded to be a model of $T_B$. A small theorem shows us how strong the concept of interpretability is:

**Theorem 1 (Interpretability of formulas).** *Assume $T_B$ interprets $T_A$. Then for any $\phi \in \mathcal{L}_\mathcal{A}$,*

$$T_A \models \phi \iff T_B \models \phi^I \tag{6}$$

*Proof. We include the lemma:*

***Lemma 1 ([Rabin, 1965]).*** *Given any $I : \mathcal{L}_\mathcal{A} \to \mathcal{L}_\mathcal{B}$, for any such $\phi \in \mathcal{L}_\mathcal{A}$, and structure $\mathfrak{B}$ of $_\mathcal{B}$,*

$$\mathfrak{B} \models \phi^I \iff \mathfrak{B}^I \models \phi. \tag{7}$$

*This lemma is proved by induction on the complexity of formulas.*

*$\rightarrow$: Assume $T_A \models \phi$. Let $\mathfrak{B} \models T_B$, and try to show $\mathfrak{B} \models \phi^I$. By the definition of interpretability, there is an $\mathfrak{A}' \models T_A$ such that $\mathfrak{B}^I \cong \mathfrak{A}'$. Hence $\mathfrak{B}^I \models T_A$, which means that $\mathfrak{B}^I \models \phi$. By the lemma $\mathfrak{B} \models \phi^I$.*

*$\leftarrow$: This time assume $T_B \models \phi^I$, let $\mathfrak{A} \models T_A$, and show $\mathfrak{A} \models \phi$. By interpretability there is a $\mathfrak{B}' \models T_B$ such that $\mathfrak{A} \cong \mathfrak{B}'^I$. $\mathfrak{B}' \models \phi^I$ and by the lemma this means $\mathfrak{B}'^I \models \phi$. And then from the isomorphism we can deduce $\mathfrak{A} \models \phi$.*

Hence if $T_B$ interprets $T_A$, for any formula $\phi \in \mathcal{L}_{\mathcal{A}}$, we can use $T_B$ to find out whether $T_A \models \phi$ simply by checking if the interpreted formula $\phi^I$ holds in $T_B$. Intuitively, the function $I$ interprets a simple theory $T_A$ in a more complicated on $T_B$; $T_B$ contains the concepts necessary to express the concepts formalized in $T_A$.

We can now define the simplest form of theory approximation. We can say that a theory $T_A$ in language $\mathcal{L}_{\mathcal{A}}$ *approximates* another theory $T_B$ in $\mathcal{L}_{\mathcal{B}}$, written $T_A <_{approx} T_B$, if $T_A$ is interpretable in $T_B$ but not the other way around. The idea is that $T_B$ is powerful enough to model $T_A$, but $T_A$ cannot do the same for $T_B$, so it must be inherently more approximate (lost information).

This method of syntactic interpretations gives us a framework in which to relate different theories with different languages through the interpretation function $I$. In order to make further distinctions in different kinds of approximation we believe it will be necessary to study the fine structure of the function $I$. $I$ may be related to the *simplifying assumptions* mentioned in [Nayak and Levy, 1995].

Some facts to notice before we continue with some examples: Assume $T_B$ interprets $T_A$. If $T_B$ were complete, then it has only one model $\mathfrak{B}$, and (2) means that all the models of $T_A$ are isomorphic to each other. Therefore $T_A$ is categorical, and complete as well. If $T_B$ is complete, then any approximation to it, according to this definition will have to be as well. On the other hand, if $T_A$ were complete, then every model of $T_B$ will have to agree when back-translated to $\mathcal{L}_{\mathcal{A}}$.

### 4.1 The Treatment of Functions and Constants under Approximation

This section illustrates how functions and constants in $\mathcal{L}_{\mathcal{A}}$ are transformed under the function $I$, and where they appear in the models of $T_A$ versus models of $T_B$. Assume $f$ is a unary function symbol of $\mathcal{L}_{\mathcal{A}}$, used in $T_A$. We can construct a predicate $P_f(x,y) \iff_{def} f(x) = y$ which represents the *graph* of $f$, and have a new language $\mathcal{L}'_{\mathcal{A}} = \mathcal{L}_{\mathcal{A}} - \{f\} + \{P_f\}$. Also we alter our theory $T'_A \equiv T_A|_{f(x)=y \leftarrow P_f(x,y)} \wedge (\forall x)(\exists y)(\forall z)[P_f(x,z) \iff y = z]$. Then the translation would be:

$$\begin{aligned}((\forall x)(\exists y)(\forall z)[P_f(x,z) \iff y = z])^I = \\ (\forall x \in \phi_=)(\exists y \in \phi_=)(\forall z \in \phi_=)[\phi_{P_f}(x,z) \iff y = z].\end{aligned} \tag{8}$$

By Theorem 1, we know that $T_B \models (8)$, so that $\phi_{P_f}$ defines within $T_B$ a function (call it $g$), which bears a relation to $f$ through the transformation from $P_f$ and $\phi_{P_f}$. Also, $g$ is only defined on the extension of $\phi_=$.

Since a constant is a 0-ary function, we can recycle the exposition above to show that our graph of $c(x) = c$ is $P_c(y)$, and (8) will assert that $P_c$ is the predicate uniquely true of $c$. Then under $I$ we will get another predicate $\phi_{P_c}$ which will be uniquely true of some other element $d \in \phi_{=}$. Intuitively then we can consider $I$ to also map objects and functions, where we denote $d = c^I$, and $g = f^I$, which are related through the predicate transformations.

### 4.2 Theory Approximation Example: Generalizing Conditionals

Consider the theory $T_B \equiv \Psi \wedge (\forall x)[P(x) \wedge \alpha(x) \implies \beta(x)]$, where $P$ is a complicated predicate we would like to ignore, and $\Psi$ some set of sentences which does not mention $P$. Can we approximate it by the much simpler theory $T_A = \Psi \wedge (\forall x)[\alpha(x) \implies \beta(x)]$?

An obvious interpretation is if $[P(x)]^I = \phi_{=}(x)$ – that is, $T_A$ can be interpreted in any model of $T_B$ provided we restrict the domain of discourse to objects obeying $P(x)$. There could be other interpretations that depend on the structure of $\alpha, \beta$ and $\Psi$. The details are omitted here.

### 4.3 Theory Approximation Example: Blocksworld

Let $T_{BW}$ be the standard situation calculus theory of Blocksworld, using the language $\mathcal{L}_{\mathcal{BW}} = (On(x, y, s), move(x, y, z), Table, A, B, C, Result)$ where $On(x, y, s)$ is the predicate stating block $x$ is on block $y$, $move(x, y, z)$ is the action that moves $x$ from $y$ to the top of $z$, $Table$ denotes a table of infinite capacity, and $A$, $B$, and $C$ are names for unique blocks. $Result$ is the standard successor function used in situation calculus. We assume $T_{BW}$ has successor state axioms to completely specify the effects of every action; if an action is not possible, we assume the world stays as it is. As we mentioned before, this theory is so simple it does not even model the physical aspect of the blocks. Another important approximation is that the table has infinite capacity. Finally, blocks are either on, or off another block – there is no concept of them being partially on a block, or being on two blocks at the same time.

Now consider a more realistic theory $T'_{BW}$ of Blocksworld that models the same blocks as in $T_{BW}$, but as roughly parallelpieds, each with a center of gravity. If the center of gravity of any tower of blocks is not supported from below, this will generate torque about this axis and cause the blocks to fall to the table. The language $\mathcal{L}'_{\mathcal{BW}}$ for such a descriptive theory includes symbols such as: $On(x, y, \delta, s)$, where $\delta$ is the deviation of the center of gravity of $x$ from $y$, and $move(x, y, z, \delta)$, which is the action of moving $x$ from $y$ to $z$ with $\delta$ deviation of $x$'s center of gravity from $z$'s. Other than the constants $Table$, $A$, $B$, and $C$, are functions $cg(x)$ which gives the center of the gravity of the tower of blocks above and including $x$, and normal arithmetic functions. Of course we need some function $surface(y, s)$, which returns some structure delineating the surface of $y$ (so that we can check if a tower of blocks is not supported and will fall).

We argue that $T_{BW} <_{approx} T'_{BW}$. To show this, we must show that $T'_{BW}$ interprets $T_{BW}$, but not the other way around. Consider the interpretation $I$

which interprets $On(x, y, s)$ as the formula $(\exists\delta)On(x, y, \delta, s)$, and $move(x, y, z) = a$ as $(\exists\delta)[a = move(x, y, z, \delta) \wedge \delta + cg(x) \in surface(z, s)]$. We also set $\phi_{=}(x)$ to be true only for the blocks and the table, and for actions which place blocks precisely over the center of gravity of other blocks. Each block $A, B, C$ is matched to the corresponding block in $T'_{BW}$, and the $Table$ to the $Table'$. The idea behind this interpretation is that models of $T_{BW}$ should correspond to "rounded-off" versions of models of $T'_{BW}$.

Take any model $\mathfrak{M}$ of $T_{BW}$. We must show there is a corresponding $\mathfrak{M}'$ of $T'_{BW}$ such that ${\mathfrak{M}'}^I$ and $\mathfrak{M}$ are isomorphic. We construct a model $\mathfrak{M}'$ of $T'_{BW}$ by taking $|\mathfrak{M}'| \supseteq |\mathfrak{M}|$. Then for each $\langle x, y, s\rangle \in On^{\mathfrak{M}}$, we put $\langle x, y, 0, s\rangle \in On^{\mathfrak{M}'}$. If $a = move^{\mathfrak{M}}(x, y, z)$, then we set $s = move^{\mathfrak{M}'}(x, y, z, 0)$. Clearly by definition $\mathfrak{M} = {\mathfrak{M}'}^I$, since $|\phi_{=}| = |\mathfrak{M}|$.

For the second condition, it is enough to show that for any model $\mathfrak{M}'$ of $T'_{BW}$, ${\mathfrak{M}'}^I \models T_{BW}$. $\phi_{=}$ in this case will make sure $|{\mathfrak{M}'}^I|$ consists only of blocks, the table, and precise moves. Then every "inexact" placement of blocks $On(x, y, \delta, s)$ is rounded down to an exact one in ${\mathfrak{M}'}^I$, and every "inexact" movement $move(x, y, z, \delta)$ becomes the exact $move(x, y, z)$.

On the other hand, $T_{BW}$ does not interpret $T'_{BW}$. $T'_{BW}$ is simply more expressive. Formally, we can show this by considering the contrapositive of Theorem 1, by taking an arbitrary $I : \mathcal{L}'_{\mathcal{BW}} \to \mathcal{L}_{\mathcal{BW}}$, and finding a $\phi$ which does not translate over. The trick is to have $\phi$ formalize one of the differences between $T'_{BW}$ and $T_{BW}$. For example, $\phi$ could be the statement that there exists a move where the block falls to the table instead of reaching its destination (because it is placed haphazardly on another block). $T'_{BW} \models \phi$. But there is no way to translate this to an expression in $\mathcal{L}_{\mathcal{BW}}$ which talks about the mysterious failure of an action, without contradicting $T_{BW}$.

### 4.4 Related Work

[Nayak and Levy, 1995] uses the same mathematical framework of syntactic interpretation to characterize *model increasing (MI) abstractions*. MI abstractions have the advantage over the syntactic ones in [Giunchiglia and Walsh, 1992] in that they capture more of the "underlying justification that leads to the abstraction," [Nayak and Levy, 1995]. Among other insights, the work shows how the ABSTRIPS abstraction is an MI one. [Levy, 1994] formalizes *irrelevance* of clauses with respect to queries on knowledge bases, as well as independence of predicate arguments. [Nayak, 1994] combines abstractions within the theory of contexts.

## 5 Rich and Poor Objects

A *rich* object is one that cannot be completely described, while a *poor* one can. At first, one possible criterion for "description" may be identifying properties, the set of properties which uniquely specify the object. This is opposed to *all* of the properties which are true of the object. The number 0, even though there

are infinitely many things to be said of it ($0 < 1, 0 < 2, 0 < 3, 0 < 4, \ldots$), can be concisely identified as the unique number which has no predecessor, and therefore is poor. On the other hand, the author may be uniquely identified, by the poor description "the person writing this paper," but the author is a rich object, not a poor one.

Whether an object is rich or poor also depends on how it is formalized, and what language is used to represent it. Situations, defined as a "snapshot of the world," are always treated as rich objects, while states, a finite collection of variables' values, are poor. A situation is rich because we can always find another property that specifies the situation further, while once we decide upon $n$ variable values, a state is completely specified and very unmysterious. But then if $n$ were very large, and only a small part of the state's variable values were ever contemplated in a theory, it might as well be a rich object.

A possible definition of a rich object is one for which we can always extend a theory of it to one which ascribes more untrivial properties. If $o_R$ is a rich object in theory $T$, then we can imagine a more expressive theory $T'$ where $T <_{approx} T'$ in which $o_R^I$ appears. Any concept true of $o_R$ in $T$ will have its interpretation true of $o_R^I$ in $T'$ by Theorem 1, and if the interpretation $I$ is injective, the distinct properties true of $o_R$ can only increase. But to guarantee that we find out more interesting facts about $o$ (and not some other object in $T'$) we will have to require something like that the $tp_{T'}(o_R^I)$, the *type* of $o_R^I$, defined as the set of all 1-place formulas of $T'$ that are true of $o_R^I$, is not interpretable in $T$.

A *rich* object $o_R$ in theory $T_0$ then, would be one for we can continually de-approximate theories about $o_R$: there is an infinite sequence of theories such that $T_0 <_{approx} T_1 <_{approx} T_2 <_{approx} T_3 \ldots$. Therefore an object $o_P$ is *poor* iff every such sequence of theories bottoms out; we run out of things to say. One object $o$ is *richer* than another $o'$ if the longest sequence of $o'$ theories is less than the longest one of $o$'s, starting from the simplest theory $o = o$ or $o' = o'$. Back to the state example, a state with $10^7$ variables is richer than one with 3, but both are poor, and can't compare to a situation.

We could define reality $\mathbf{R}$ as the class of models which bound every such de-approximating sequence of theories: for every extension of a theory in $\mathcal{L}_{\mathcal{O}}$, the supremum of the sequence is a theory $T_{sup}$ whose models are contained in $\mathbf{R}$.

## 6 Back to the Objects

In §3 we presented a mathematical definition for what it meant for one theory to approximate another. Now we return to our task of formalizing each of the three types of approximate objects within the theory. The formalization of a theory approximation is needed for object approximation because as we noted earlier, whether an object is approximate depends on its context, which refers to how it is described and what language is used to describe it.

In the description of these types of objects in §1.4, we compared each object in some given theory (assumed to be in language $\mathcal{L}_{\mathcal{O}}$) to its counterpart in reality in $\mathcal{L}_{\mathcal{R}}$. Below instead of talking about theories in $\mathcal{L}_{\mathcal{O}}$ and $\mathcal{L}_{\mathcal{R}}$, we generalize the discussion to theories $T_A$ and $T_B$, where $T_A <_{approx} T_B$. Thus the definitions below will apply to any pair of theories.

### 6.1 Approximate Objects of Type I (Poorer Version of a Richer Object)

Let $c$ be an object constant, in theory $T_A$, of type I. This means that there exists a more precise theory $T_B$ with corresponding object $c^I$, for which $c$ is but a poor approximation. This is represented by three facts:

1. $T_A <_{approx} T_B$.
2. $c$ has some corresponding object, $c^I$ in $T_B$.
3. The description of $c$ in $T_A$ is poorer than that of $c^I$ in $T_B$.

To implement type I objects all we require is a de-approximation $T_B$ to the current theory $T_A$. The existence of $c^I$ will be guaranteed by the interpretation function $I$, and since generally $I$ is injective, will map different properties in $T_A$ to different ones in $T_B$, so that only more things can be said of $c^I$ in $T_B$. Naturally, we must add the constraints given in §5 about $tp_{T_B}(c^I)$ not being expressible in $T_A$ to make sure its description gets richer.

If $T_B = Th_{\mathcal{L}_{\mathcal{R}}}(\mathbf{R})$, almost every object $o$ we imagine would be approximate, since every theory $T$ of $o$ can be extended to $Th_{\mathcal{L}_{\mathcal{R}}}(\mathbf{R})$ (it being a $<_{approx}$ upper bound on all theories), and we can always imagine $o$'s richer analog in reality.

### 6.2 Approximate Objects of Type II (Objects with no Basis in Reality)

If an object (or concept) $c$ has no basis in reality, then of course an interpretation $I$ cannot be built, since $c$ maps to some concept $c^I$ in $\mathfrak{R}$, which cannot exist! The color red, "what the U.S. wants," corners, baldness, and heaps are all examples. Although these concepts have no immediate corresponding object in $\mathbf{R}$, they do seem to correspond to some composition of objects in reality. For example, "red" is the sensation of viewing light of wavelengths from 600 to 700 nanometers. Similarly, "what the U.S. wants" is some complicated array of what the President, U.S. diplomats, populace, etc. desire. A corner is some spatial extent about a physical corner, while baldness is some collection of states of a person with very little hair. The objects are approximate not only because they correspond to a composition of objects, but the nature of composition is itself vague.

For many of these examples, the approximation is an association of one object in our base theory $T_A$ with a *collection* of objects in a more precise theory $T_A <_{approx} T_B$. For example the concept of a *heap* in $T_A$ is associated with some set of actual heaps in $T_B$. We need to have an altered interpretation function that maps objects in $T_A$ to sets in $T_B$. The epistemology of this is not worked out yet and we hope to say much more later.

### 6.3 Approximate Objects of Type III (Counterfactual Objects)

[McCarthy, 2000] introduces a function $Whatif?(p, x)$, where $p$ is a proposition, and $x$ some constant symbol. It can be used to consider counterfactual concepts such as "What would have happened if another car had come over the hill when you passed that Mercedes." We can adapt this interesting function and consider $Whatif?(p, T)$, where $T$ is a first-order theory about the world and $p$ still a proposition, both in the same language $\mathcal{L}$. $Whatif?(p, T)$ would be the resulting theory if $p$ were to hold.

Clearly, $Whatif?(p, T) \models p$. Also, if $T \models p$, then $Whatif?(p, T) = T$. If this isn't the case, $Whatif?(p, T)$ can be imagined as some minimal re-arrangement of $T$ that would be consistent with $p$. $Whatif?$ could be implemented using the mechanisms in [Costello and McCarthy, 1999]. For non-trivial $p$ and $T$, $Whatif?(p, T)$ could be infinitely refinable, which means that the function itself would be rich, and any finite theory it returned would only be some approximation! To try to keep this from happening we assume that $Whatif?(p, T)$ returns a theory also in language $\mathcal{L}$. Hence the more expressive $\mathcal{L}$ is, the more interesting $Whatif?(p, T)$ will be.

$Whatif?(p, T)$, being a counterfactual theory, will reference objects of type III, which won't exist in $T$, as in "the car that came over the hill." The properties of these type III objects should be limited to those ascribed to them by $p$, along with whatever ramifications that may follow from $Whatif?(p, T)$.

## 7 When an Approximate Theory Works in Reality

Consider a robot stacking blocks. Suppose all the blocks are very close to being idealized cubes, and the servo-mechanism in the robot arm is programmed so carefully that the robot always stacks blocks precisely without error. Then to the robot, the typical theory of Blocksworld will be a true description of reality, even though we smarter humans know that it is a highly idealized description of the world that won't work in general.

But to the robot, this theory is correct. We would like to define under what circumstances a simple approximate theory will work correctly in that complex reality, because then we will know when we can get away with such approximations. This may also give us hints as how to parameterize a theory with *Ab*s so that it will be more robust outside the boundaries of approximation, and also more elaboration tolerant.

We define this idea of when an approximate theory correctly predicts reality as *coherence*. A theory $T$ is *coherent* with respect to a more realistic theory $T'$ and restrictions (same as simplifying assumptions) $\Phi$ if: $I$ is an interpretation from $\mathcal{L}$ to $\mathcal{L}'$ and $T' \wedge \Phi \models T^I$, that is, every model of $T' \wedge \Phi$ will also model $T$ interpreted in $T'$. If $T <_{approx} T'$ then coherence follows. Coherence is *Lucky*-ness for first order theories.

There are already examples of coherent theories in this paper. In §4.2, if we restrict our domain of discourse to be $\{x \mid P(x)\}$ we can approximate $\Psi \wedge$

$(\forall x)[P(x) \wedge \alpha(x) \implies \beta(x)]$, with $\Psi \wedge (\forall x)[\alpha(x) \implies \beta(x)]$. This is obvious, but we can imagine more complicated theories for which we would need the formalism to deduce when approximations would apply. If $P(x) = \neg Ab(x)$ we could apply the theory in reverse to find out how to add *Ab*s to a theory to make it more elaboration tolerant.

## 8 Conclusions

This paper is very preliminary and many of the definitions of approximation are yet only approximations themselves. Much more work needs to be done in order to make our theories precise. We need to further explore the structure of the interpretation functions $I$ and understand how they map concepts in one theory to those in another, especially with regard to approximate objects of type II. This will require an understanding of how to ground symbols. Perhaps we can develop this understanding by considering the approximation of $T_A <_{approx} T_B$ only within the context of how they relate to reality **R**.

Some other formal ideas from mathematics may be relevant to the properties of epistemologically rich and poor objects. For example forcing and generic sets [Feferman, 1965] formalize the notion that a concept can be described in some finite set of sentences, which could be a better criterion for whether an object is poor. The theory sequences we use to determine richness/poorness might be related to the sequence of finite sets of conditions $Q_0, Q_1, \ldots$.

At least this theory explains why there are so many different AI theories of action and change. Each formalism is just a different approximation to reality. $<_{approx}$ is not a total order, so many different, incomparable approximations are possible. Also, an initial review of [Fine, 1985] suggests that *arbitrary objects* are a kind of approximate object that would be fruitful to study.

**Acknowledgments.** We are thankful for guidance from John McCarthy, as well as Rada Chirkova and Sheila McIlraith for pointers to additional work in abstraction. We are also grateful to the anonymous referees for their helpful comments and suggestions.

## References

[Alston, 1967] Alston, W. P. (1967). Vagueness. In Edwards, P., editor, *Encyclopedia of Philosophy*, volume 8, pages 218–221. MacMillan.

[Baudisch et al., 1985] Baudisch, A., Seese, D., Tuschik, P., and Weese, M. (1985). *Model-Theoretic Logics*, chapter Decidability and Quantifier Elimination, pages 235–268. Springer-Verlag.

[Broome, 1984] Broome, J. (1984). Indefiniteness in identity. *Analysis*, 44:6–12.

[Costello and McCarthy, 1999] Costello, T. and McCarthy, J. (1999). Useful Counterfactuals[6]. *Electronic Transactions on Artificial Intelligence*.

[Dreyfus and Dreyfus, 1984] Dreyfus, H. and Dreyfus, S. (1984). From Socrates to Expert Systems: The Limits of Calculative Rationality[7].

[6] http://www-formal.stanford.edu/jmc/counterfactuals.html

[7] http://socrates.berkeley.edu/~hdreyfus/html/paper_socrates.html

[Feferman, 1965] Feferman, S. (1965). Some applications of the notions of forcing and generic sets. *Fundamenta Mathematicae*, 56:325–345.

[Fine, 1985] Fine, K. (1985). *Reasoning with Arbitrary Objects*. Aristotelian Society Series. Oxford.

[Giunchiglia and Walsh, 1992] Giunchiglia, F. and Walsh, T. (1992). A theory of abstraction. *Artificial Intelligence*, 57(2-3):323–389.

[Goguen, 1968] Goguen, J. A. (1968). The logic of inexact concepts. *Synthese*, 19:325–373.

[Hobbs, 1985] Hobbs, J. R. (1985). Granularity. In *International Joint Conference on Artificial Intelligence (IJCAI'85)*, pages 432–435.

[Howe, 1994] Howe (1994). Rich Object[8]. In Howe, D., editor, *Free Online Dictionary of Computing*.

[Levy, 1994] Levy, A. Y. (1994). Creating abstractions using relevance reasoning. In *AAAI, Vol. 1*, pages 588–594.

[McCarthy, 1979] McCarthy, J. (1979). Ascribing mental qualities to machines[9]. In Ringle, M., editor, *Philosophical Perspectives in Artificial Intelligence*. Harvester Press. Reprinted in [**?**].

[McCarthy, 1998] McCarthy, J. (1998). Elaboration Tolerance[10]. In *In Proceedings of the Fourth Symposium on Logical Formalizations of Common Sense Reasoning*.

[McCarthy, 2000] McCarthy, J. (2000). Approximate objects and approximate theories. In Cohn, A. G., Giunchiglia, F., and Selman, B., editors, *KR2000: Principles of Knowledge Representation and Reasoning,Proceedings of the Seventh International conference*, pages 519–26. Morgan-Kaufman.

[McCarthy and Hayes, 1969] McCarthy, J. and Hayes, P. J. (1969). Some Philosophical Problems from the Standpoint of Artificial Intelligence[11]. In Meltzer, B. and Michie, D., editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press.

[Morgenstern, 1998] Morgenstern, L. (1998). Common Sense Problem Page[12]. Webpage.

[Nayak, 1994] Nayak, P. P. (1994). Representing multiple theories. In Hayes-Roth, B. and Korf, R., editors, *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1154–1160, Menlo Park, CA. AAAI Press.

[Nayak and Levy, 1995] Nayak, P. P. and Levy, A. (1995). A semantic theory of abstractions. In Mellish, C., editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 196–203, San Francisco. Morgan Kaufmann.

[Rabin, 1965] Rabin, M. O. (1965). A simple method for undecidability proofs and some applications. In Bar-Hillel, Y., editor, *Logic, Methodology and Philosophy of Science*, pages 58–68. North Holland Publishing Company.

[Sorensen, 1997] Sorensen, R. (1997). Vagueness[13]. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. The Metaphysics Research Lab at the Center for the Study of Language and Information, Stanford University, Stanford, CA.

[8] http://burks.brighton.ac.uk/burks/foldoc/83/99.htm
[9] http://www-formal.stanford.edu/jmc/ascribing.html
[10] http://www-formal.stanford.edu/jmc/elaboration.html
[11] http://www-formal.stanford.edu/jmc/mcchay69.html
[12] http://www-formal.stanford.edu/leora/cs/
[13] http://plato.stanford.edu/entries/vagueness/

[Tarski et al., 1953] Tarski, A., Mostowski, A., and Robinson, R. M. (1953). Undecidable theories. In *Studies in logic and the foundations of mathematics*. North-Holland Pub. Co., Amsterdam.

[Tye, 2000] Tye, M. (2000). Vagueness[14]. In *Routledge Encyclopedia of Philosophy*.

[Williamson, 1994] Williamson, T. (1994). *Vagueness*. Routledge.

[14] http://www.rep.routledge.com/philosophy/

# Model Minimization in Hierarchical Reinforcement Learning

Balaraman Ravindran and Andrew G. Barto

Department of Computer Science,
University of Massachusetts,
Amherst, MA 01003, USA
{ravi,barto}@cs.umass.edu

**Abstract.** When applied to real world problems Markov Decision Processes (MDPs) often exhibit considerable implicit redundancy, especially when there are symmetries in the problem. In this article we present an MDP minimization framework based on homomorphisms. The framework exploits redundancy and symmetry to derive smaller equivalent models of the problem. We then apply our minimization ideas to the options framework to derive relativized options—options defined without an absolute frame of reference. We demonstrate their utility empirically even in cases where the minimization criteria are not met exactly.

## 1 Introduction

Researchers in artificial intelligence (AI) and in particular machine learning (ML) have long recognized that extending AI and ML approaches to more complex real-world domains requires incorporating the ability to handle and form various abstractions, both temporal and spatial. In this article we present a Markov decision processes (MDP) minimization framework we developed earlier [15] that allows us to abstract away redundancy in the problem definition. We then apply these ideas to hierarchical Reinforcement Learning (RL). Our framework is an extension of a MDP minimization framework developed by Dean and Givan [4, 6].

Model minimization methods attempt to abstract away redundancy in an MDP model and derive an "equivalent" smaller model. To illustrate model minimization, consider the simple gridworld shown in Figure 1(a). The goal state is labelled G. The gridworld is symmetric about the NE-SW diagonal. Hence taking action E in state A is equivalent to taking action N in state B, in the sense that they go to equivalent states that are one step closer to the goal. One can say that the state-action pairs (A, E) and (B, N) are equivalent. We can exploit this notion of equivalence to construct a smaller model of the gridworld, one that can be used to derive a solution to the original problem. Such a reduced gridworld is shown in Figure 1(b).

We base our approach to MDP minimization on the notion of *MDP homomorphisms*. This is an extension of machine homomorphisms from finite state

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 196–211, 2002.

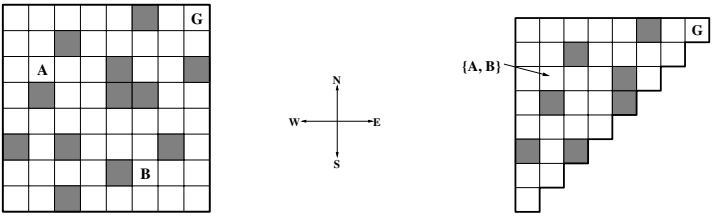



**Fig. 1.** (a) A symmetric gridworld problem. The goal state is $G$ and there are four deterministic actions. States $A$ and $B$ are equivalent in the sense described in the text. (b) A reduced model of the gridworld in (a). The states $A$ and $B$ in the original problem correspond to the single state $\{A, B\}$ in the reduced problem. A solution to this reduced gridworld can be used to derive a solution to the full problem.

automata (FSA) literature [9]. We extend the notion to MDPs by incorporating decision making and stochasticity. The key novelty in our approach is the extension of notions of equivalence to state-action pairs. This enables us to apply our results to a wider class of problems and extend existing MDP minimization frameworks in ways not possible earlier. Specifically, by employing group theoretic concepts we show that our extended minimization framework can abstract away symmetries in an MDP model.

The minimization framework we develop for MDPs can be employed readily by RL algorithms for spatial abstraction. The options framework [17] enables RL algorithms to employ temporal abstractions in the form of temporally extended actions, or *options*. Extending our algebraic framework to a hierarchical RL setting such as the options framework opens up additional possibilities. In this paper we introduce *relativized options*, an extension to the options framework based on "partial" MDP homomorphisms that allows us to define option policies without an absolute frame of reference and hence widens the applicability of an option, enables greater knowledge transfer across tasks and more efficient use of experience. We also investigate employing relativized options in cases where the abstraction conditions are not satisfied exactly. We introduce approximate homomorphisms that model such scenarios using the notion of bounded-parameter MDPs [7].

In the next section we present some notation we will be using. In Section 3 we outline our model minimization framework and state some results. We also show how our framework can exploit symmetries of MDPs. In Section 4 we introduce relativized options and present some experimental results. In Section 5 we define approximate homomorphisms and empirically demonstrate their usefulness. We conclude with a discussion on related work and some future directions of research.

## 2 Notation

A *Markov Decision Process* is a tuple $\langle S, A, \Psi, P, R \rangle$, where $S$ is a finite set of states, $A$ is a finite set of actions, $\Psi \subseteq S \times A$ is the set of admissible state-action pairs, $P : \Psi \times S \to [0, 1]$ is the transition probability function with $P(s, a, s')$ being the probability of transition from state $s$ to state $s'$ under action $a$, and $R : \Psi \to \mathbb{R}$ is the expected reward function, with $R(s, a)$ being the expected reward for performing action $a$ in state $s$. We assume that the rewards are bounded. Let $A_s = \{a | (s, a) \in \Psi\} \subseteq A$ denote the set of actions admissible in state $s$. We assume that for all $s \in S$, $A_s$ is non-empty. A *stochastic policy* $\pi$ is a mapping from $\Psi$ to the real interval $[0, 1]$ with $\sum_{a \in A_s} \pi(s, a) = 1$ for all $s \in S$. For any $(s, a) \in \Psi$, $\pi(s, a)$ gives the probability of picking action $a$ in state $s$. The solution of an MDP is an *optimal policy* $\pi^\star$ that uniformly dominates all other possible policies for that MDP.

Let $B$ be a partition of a set $X$. For any $x \in X$, $[x]_B$ denotes the block of $B$ to which $x$ belongs. Any function $f$ from a set $X$ to a set $Y$ induces a partition $B_f$ on $X$, with $[x]_{B_f} = [x']_{B_f}$ if and only if $f(x) = f(x')$. Let $B$ be a partition of $Z \subseteq X \times Y$, where $X$ and $Y$ are arbitrary sets. The *projection of $B$ onto $X$* is the partition $B|X$ of $X$ such that for any $x, x' \in X$, $[x]_{B|X} = [x']_{B|X}$ if and only if every block of $B$ containing a pair in which $x$ ($x'$) is a component also contains a pair in which $x'$ ($x$) is a component. A *partition of an MDP* $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$ is a partition of $\Psi$. Given a partition $B$ of $\mathcal{M}$, the *block transition probability of* $\mathcal{M}$ is the function $T : \Psi \times B|S \to [0, 1]$ defined by $T(s, a, [s']_{B|S}) = \sum_{s'' \in [s']_{B|S}} P(s, a, s'')$. In other words, $T(s, a, [s']_{B|S})$ is the probability of transiting from state $s$ to some state in the block $[s']_{B|S}$ (i.e. the block to which state $s'$ belongs) under action $a$.

An option (or a temporally extended action) [17] in an MDP $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$ is defined by the tuple $O = \langle \mathcal{I}, \pi, \beta \rangle$, where the initiation set $\mathcal{I} \subseteq S$ is the set of states in which the option can be invoked, $\pi$ is the option policy, and the termination function $\beta : S \to [0, 1]$ gives the probability of the option terminating in any given state. The option policy can in general can be a mapping from arbitrary sequences of state-action pairs (or histories) to action probabilities.

## 3 MDP Homomorphisms

In this article we present a formalism that captures the intuitive notion of equivalence illustrated in Figure 1. In Figure 1(a), we consider states A and B equivalent, since for every action in A that puts you in a state a certain distance from the goal, there is an action in B that takes you to an equivalent state at the same distance from the goal. More generally, in an MDP $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$ we consider state $s_1$ equivalent to state $s_2$ if for every action available in $s_1$, there is some action in $s_2$ that results in similar behavior with respect to the transition structure $P$ and vice versa. We also require that the actions be equivalent with respect to the reward function $R$. We can then derive a simpler model $\mathcal{M}'$ of $\mathcal{M}$ by aggregating together *blocks* of equivalent states. In other words, $\mathcal{M}'$ is a

simpler model of $\mathcal{M}$ if there exists a transformation from $\mathcal{M}$ to $\mathcal{M}'$ that preserves the transition and reward structure and maps equivalent states in $\mathcal{M}$ to the same state in $\mathcal{M}'$, and equivalent actions in $\mathcal{M}$ to the same action in $\mathcal{M}'$. An MDP homomorphism from $\mathcal{M}$ to $\mathcal{M}'$ is such a transformation. Formally, we define it as:

**Definition:** An *MDP homomorphism* $h$ from an MDP $\mathcal{M} = \langle S, A, \Psi, P, R\rangle$ to an MDP $\mathcal{M}' = \langle S', A', \Psi', P', R'\rangle$ is a surjection from $\Psi$ to $\Psi'$, defined by a tuple of surjections $\langle f, \{g_s | s \in S\}\rangle$, with $h((s,a)) = (f(s), g_s(a))$, where $f : S \to S'$ and $g_s : A_s \to A'_{f(s)}$ for $s \in S$, such that:

$$P'(f(s), g_s(a), f(s')) = T(s, a, [s']_{B_h|S}),\ \forall s, s' \in S, a \in A_s \tag{1}$$

$$R'(f(s), g_s(a)) = R(s, a),\ \forall s \in S, a \in A_s \tag{2}$$

We call $\mathcal{M}'$ the *homomorphic image* of $\mathcal{M}$ under $h$. We use the shorthand $h(s,a)$ to denote $h((s,a))$. From condition (1) we can see that state-action pairs that have the same image under $h$ have the same block transition behavior in $\mathcal{M}$, i.e., the same probability of transiting to any given block of states with the same image under $f$. Condition (2) says that state-action pairs that have the same image under $h$ have the same expected reward. This definition of a MDP homomorphism leads to the following definition of equivalence of states and state-action pairs.

**Definition:** State action pairs $(s_1, a_1)$ and $(s_2, a_2) \in \Psi$ are *equivalent* if there exists a homomorphism $h$ of $\mathcal{M}$ such that $h(s_1, a_1) = h(s_2, a_2)$. States $s_1$ and $s_2 \in S$ are *equivalent* if i) for every action $a_1 \in A_{s_1}$, there is an action $a_2 \in A_{s_2}$ such that $(s_1, a_1)$ and $(s_2, a_2)$ are equivalent, and ii) for every action $a_2 \in A_{s_2}$, there is an action $a_1 \in A_{s_1}$, such that $(s_1, a_1)$ and $(s_2, a_2)$ are equivalent.

Thus the surjection $f$ maps equivalent states of $\mathcal{M}$ onto the same image state in $\mathcal{M}'$, while $g_s$ is a *state dependent* mapping of the actions in $\mathcal{M}$ onto image actions in $\mathcal{M}'$. For example, if $h = \langle f, \{g_s | s \in S\}\rangle$ is a homomorphism from the gridworld of Figure 1(a) to that of Figure 1(b), then $f(A) = f(B)$ is the state marked $\{A, B\}$ in Figure 1(b). Also $g_A(E) = g_B(N) = E$, $g_A(W) = g_B(S) = W$, and so on. A policy in $\mathcal{M}'$ *induces* a policy in $\mathcal{M}$ and the following describes how to derive such an induced policy.

**Definition:** Let $\mathcal{M}'$ be an image of $\mathcal{M}$ under homomorphism $h = \langle f, \{g_s | s \in S\}\rangle$. For any $s \in S$, $g_s^{-1}(a')$ denotes the set of actions that have the same image $a' \in A'_{f(s)}$ under $g_s$. Let $\pi$ be a stochastic policy in $\mathcal{M}'$. Then $\pi$ *lifted to* $\mathcal{M}$ is the policy $\pi_{\mathcal{M}}$ such that for any $a \in g_s^{-1}(a')$, $\pi_{\mathcal{M}}(s,a) = \pi(f(s), a') \Big/ \left|g_s^{-1}(a')\right|$.

*Note:* It is sufficient that $\sum_{a \in g_s^{-1}(a')} \pi_{\mathcal{M}}(s,a) = \pi(f(s),a')$, but we use the above definition to make the lifted policy unique.

**Theorem 1:** Let $\mathcal{M}' = \langle S', A', \Psi', P', R' \rangle$ be the image of $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$ under the homomorphism $h = \langle f, \{g_s | s \in S\} \rangle$. If $\pi^\star$ is an optimal policy for $\mathcal{M}'$, then $\pi^\star_{\mathcal{M}}$ is an optimal policy for $\mathcal{M}$.[1]

Theorem 1 establishes that an MDP can be solved by solving one of its homomorphic images. To achieve the most impact, we need to derive a smallest homomorphic image of the MDP, i.e., an image with the least number of admissible state-action pairs. The following definition formalizes this notion.

**Definition:** An MDP $\mathcal{M}$ is a *minimal MDP* if for every homomorphic image $\mathcal{M}'$ of $\mathcal{M}$, there exists a homomorphism from $\mathcal{M}'$ to $\mathcal{M}$. A *minimal image* of an MDP $\mathcal{M}$ is a homomorphic image of $\mathcal{M}$ that is also a minimal MDP.

The model minimization problem can now be stated as: "find a minimal image of the given MDP". Since this can be computationally prohibitive, we frequently settle for a reasonably reduced model, even if it is not a minimal MDP. This minimization framework extends the approach proposed by Dean and Givan [4,6]. They employ *stochastic bisimulations* [12] on state sets of MDPs and do not consider state-action equivalence. If we restrict homomorphisms to only the state set, our approach is equivalent to theirs in terms of the reductions achieved. The theoretical results established in their framework hold, with suitable modifications, in our framework also. Specifically, by incorporating our extended definitions of equivalence, we can extend their algorithm for computing minimal models of MDPs to compute minimal models as defined above. Employing state-action equivalence allows us to achieve greater reduction in model size than possible with Dean and Givan's framework. For example, the gridworld in Figure 1(a) is irreducible if we consider state equivalence alone. We also explicitly model symmetries of MDPs in our framework.

### 3.1 Symmetries of MDPs

We formalize the notion of MDP symmetries employing group theoretic concepts and show that abstracting symmetries is a special case of the minimization procedure we developed above.

**Definition:** An MDP homomorphism $h = \langle f, \{g_s | s \in S\} \rangle$ from MDP $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$ to MDP $\mathcal{M}' = \langle S', A', \Psi', P', R' \rangle$ is an *MDP isomorphism* from $\mathcal{M}$ to $\mathcal{M}'$ if and only if $f$ and $g_s$, $s \in S$, are bijective. $\mathcal{M}$ is said to be *isomorphic* to $\mathcal{M}'$ and vice versa. An MDP isomorphism from an MDP $\mathcal{M}$ to itself is an *automorphism* of $\mathcal{M}$.

Intuitively one can see that automorphisms can be used to describe symmetries in a problem specification. In the gridworld example of Figure 1, a reflection of

[1] The proofs of the various theorems are presented in ref. [15].

the states about the NE-SW diagonal and a swapping of actions N and E and of actions S and W is an automorphism. It is easy to see that this mapping captures the equivalence discussed earlier.

**Definition:**The set of all automorphisms of an MDP $\mathcal{M}$, denoted by Aut$\mathcal{M}$, forms a group under composition of homomorphisms. This group is the *symmetry group* of $\mathcal{M}$.

Let $\mathcal{G}$ be a subgroup of Aut$\mathcal{M}$ denoted by $\mathcal{G} \leq \text{Aut}\mathcal{M}$. The subgroup $\mathcal{G}$ induces a partition $B_{\mathcal{G}}$ of $\Psi$: $[(s_1, a_1)]_{B_{\mathcal{G}}} = [(s_2, a_2)]_{B_{\mathcal{G}}}$ if and only if there exists $h \in \mathcal{G}$ such that $h(s_1, a_1) = (s_2, a_2)$. Since $\mathcal{G}$ is a subgroup, this implies that there exists $h^{-1} \in \mathcal{G}$ such that $h^{-1}(s_2, a_2) = (s_1, a_1)$.

**Theorem 2:** Let $\mathcal{G} \leq \text{Aut}\mathcal{M}$ be a group of automorphisms of $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$. There exists a homomorphism $h^{\mathcal{G}}$ from $\mathcal{M}$ to some $\mathcal{M}'$, such that the partition induced by $h^{\mathcal{G}}$, $B_{h^{\mathcal{G}}}$, is the same partition as $B_{\mathcal{G}}$.

The image of $\mathcal{M}$ under $h^{\mathcal{G}}$ is called the $\mathcal{G}$-*reduced image* of $\mathcal{M}$ and if $\pi^{\star}$ is an optimal policy for some $\mathcal{G}$-reduced image of MDP $\mathcal{M}$, then $\pi^{\star}_{\mathcal{M}}$ is an optimal policy for $\mathcal{M}$. Frequently the Aut$\mathcal{M}$-reduced model of an MDP is a minimal image. We can take advantage of structure inherent in a symmetry group and the induced partition in developing efficient minimization algorithms.

Ours is not the first work to study symmetries of MDPs. Zinkevich and Balch [19] define symmetries employing equivalence relations on the state-action pairs of an MDP. They do not make connections to group theoretic concepts or to minimization algorithms. They show that the optimal action-value function of a symmetric system is symmetric and suggest that the action-value function entries be duplicated. They also study in some detail symmetries that arise in multi-agent systems.

## 4 Relativized Options

It is often the case that both conditions of a homomorphism do not hold for the entire $\Psi$ space of an MDP but only over parts of it. For example, consider the problem of navigating in the gridworld environment shown in Figure 2(a). The goal is to be in the central corridor after collecting all the objects in the world. A more complete description of the task is provided in Section 4.1. The entire gridworld as such is irreducible. But each of the rooms in the world are equivalent to one another and simple transformation such as reflections and rotations map them onto each other. Thus we can create a "partial" homomorphic image of this environment, shown in Figure 2(b), with the homomorphic conditions holding only for the states in the rooms and not in the corridor. The states in which the homomorphic conditions do not hold get mapped to a "catch all" absorbing state, shown as a dark oval. All actions from these states get mapped to an absorbing

action in the image MDP. Formally we can define a partial homomorphism as follows:

**Definition:**A *partial MDP homomorphism* from $\mathcal{M} = \langle S, A, \Psi, P, R\rangle$ to $\mathcal{M}' = \langle S' \cup \{\tau\}, A' \cup \{\alpha\}, \Psi' \cup \{(\tau, \alpha)\}, P', R'\rangle$ is a surjection from $\Psi$ to $\Psi' \cup \{(\tau, \alpha)\}$, defined by a tuple of surjections $h = \langle f, \{g_s | s \in S\}\rangle$, with $h(s, a) = (f(s), g_s(a))$, where $f : S \to S' \cup \{\tau\}$ and $g_s : A_s \to A'_{f(s)}$ for $s \in S$, such that:

$$P'(f(s), g_s(a), f(s')) = T(s, a, [s']_{B_h|S}),\ \forall s \in f^{-1}(S'), s' \in S, a \in A_s \quad (3)$$

$$P'(\tau, \alpha, \tau) = 1.0 \quad (4)$$

$$R'(f(s), g_s(a)) = R(s, a),\ \forall s \in f^{-1}(S'), a \in A_s \quad (5)$$

We call $\mathcal{M}'$ the *partial* homomorphic image of $\mathcal{M}$ under $h$. The state $\tau$ is an absorbing state in $\mathcal{M}'$ with one action $\alpha$ that transitions to $\tau$ with probability 1. The homomorphism conditions hold only for states that do not map to $\tau$. All the actions in states that map to $\tau$, map to $\alpha$. Lifting policies defined in $\mathcal{M}'$ yield policy fragments in $\mathcal{M}$, with action probabilities specified only for elements in the support of $h$, i.e., $h^{-1}(\Psi')$. In Figure 2, $\tau$ corresponds to the state represented as a black oval and $\alpha$ is indicated by the solid arrow. All state-action pairs, with the state component in the corridor, map to $(\tau, \alpha)$ under the partial homomorphism. We can extend MDP minimization algorithms to find partial homomorphic images by suitably restricting the search for homomorphisms to a subset of $\Psi$. One approach to taking advantage of partial homomorphisms is to combine our minimization framework with hierarchical learning approaches.

The *options framework* is a hierarchical learning framework introduced by Sutton, Precup and Singh [17]. Options are temporally extended actions that take multiple time steps to complete. A class of options, known as "sub-goal" options, are defined as policy fragments to achieve a certain sub-goal or accomplish a certain sub-task [14]. Frequently, sub-goal options satisfy the Markov property and the option policy is defined as a map from some subset of $\Psi$ to action probabilities. In such instances it is possible to specify the the desired sub-goal of the option and to implicitly define the option policy as the solution to an *option MDP*.

The option MDP corresponding to a sub-goal option $O$ is given by $\mathcal{M}_O = \langle S' \cup \{\tau\}, A' \cup \{\alpha\}, \Psi', P', R_O\rangle$, where $S' \subseteq S$, is the states in which the option policy needs to be defined, $\tau$, is an absorbing state representing the states in $S - S'$, $A' = A$, $\Psi' = \{(s, a) | (s, a) \in \Psi, s \in S'\} \cup \{(\tau, \alpha)\}$, $P'(s, a, s') = P(s, a, s')$, if $(s, a) \in \Psi'$, $s' \in S'$, $P'(\tau, \alpha, \tau) = 1$, and $P'(s, a, \tau) = \sum_{s' / \in S'} P(s, a, s')$ for all $(s, a)$ in $\Psi'$ and $R_O$ is a reward function chosen depending on the sub-task $O$ represents. We refer to the states in which the option policy is defined, $S'$ in this case, as the *domain* of the option. We can also learn the option policy online by learning a solution to the option MDP. Such an approach is particularly useful when sub-goals are easy to identify but developing policies to achieve such sub-goals are non-trivial.

In the gridworld in Figure 2(a), an option that accomplishes the task of collecting an object and leaving room 1 can be defined as a solution to the MDP

in Figure 2(b), with the appropriate reward function. We can define similar options for each of the rooms in the world. Formally we define a Markov sub-goal option as follows:

**Definition:** A Markov sub-goal option of an MDP $\mathcal{M}$ is defined by $O = \langle \mathcal{M}_O, \mathcal{I}, \beta \rangle$, where $\mathcal{I} \subseteq S$ is the initiation set of the option, $\beta : S \rightarrow [0,1]$, is the termination function and $\mathcal{M}_O = \langle S' \cup \{\tau\}, A' \cup \{\alpha\}, \Psi', P', R_O \rangle$ is the option MDP.

The option policy $\pi$ is obtained by solving $\mathcal{M}_O$, treating it as an episodic task [16] with the possible initial states of the episodes given by $\mathcal{I}$ and the termination of each episode determined by the option's termination function $\beta$.

As is evident, the option MDP $\mathcal{M}_O$ is a partial homomorphic image of the MDP $\langle S, A, \Psi, P, R_O \rangle$, with the blocks of the induced partition of $\Psi$ being mostly singletons. The one block that is not a singleton contains all the states not in $S'$. We can apply our minimization methods reduce the option MDP further. This allows us to abstract away redundancy in the option definition and derive a more compact definition for the option. We refer to this compact option as a *relativized option*. Such options are an extension of the notion of relativized operators introduced by Iba [10]. Formally we define a relativized option as follows:

**Definition:** A *relativized option* of an MDP $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$ is the tuple $O = \langle h, \mathcal{M}_O, \mathcal{I}, \beta \rangle$, where $\mathcal{I} \subseteq S$ is the initiation set, $\beta : S' \rightarrow [0,1]$ is the termination function and $h = \langle f, \{g_s | s \in S\} \rangle$ is a partial homomorphism from the MDP $\langle S, A, \Psi, P, R_O \rangle$ to the option MDP $\mathcal{M}_O$ with $R_O$ chosen based on the sub-task.

The option MDP is defined as $\mathcal{M}_O = \langle S' \cup \{\tau\}, A' \cup \{\alpha\}, \Psi', P', R' \rangle$, where $S' = f(S_O)$, where $S_O \subseteq S$ is the domain of $O$, $\Psi' = h(\Psi)$, $P'$ satisfies conditions (3) and (4) and $R'$ satisfies condition (5). The option policy $\pi : \Psi' \rightarrow [0,1]$ is obtained by solving $\mathcal{M}_O$ by treating it as an episodic task as before. Note that the initiation set is defined over the state space of $\mathcal{M}$ and not that of $\mathcal{M}_O$. Since the initiation set is typically used by the higher level when invoking the option, we decided to define it over $S$. When lifted to $\mathcal{M}$, $\pi$ is suitably transformed into policy fragments over $\Psi$ depending on the state of $\mathcal{M}$ the system is currently in.

Going back to our example in Figure 2(a) we can now define a single relativized option using the option MDP of Figure 2(b) that represents a option to collect the object and leave a room. The policy learned in this option MDP can then be suitable lifted to $\mathcal{M}$ to provide different policy fragments in the different rooms.

### 4.1 Illustrative Example

We now provide a complete description of the simple gridworld task in Figure 2(a) and some experimental results to illustrate the utility of relativized options. The agent's goal is to collect all the objects in the various rooms by occupying

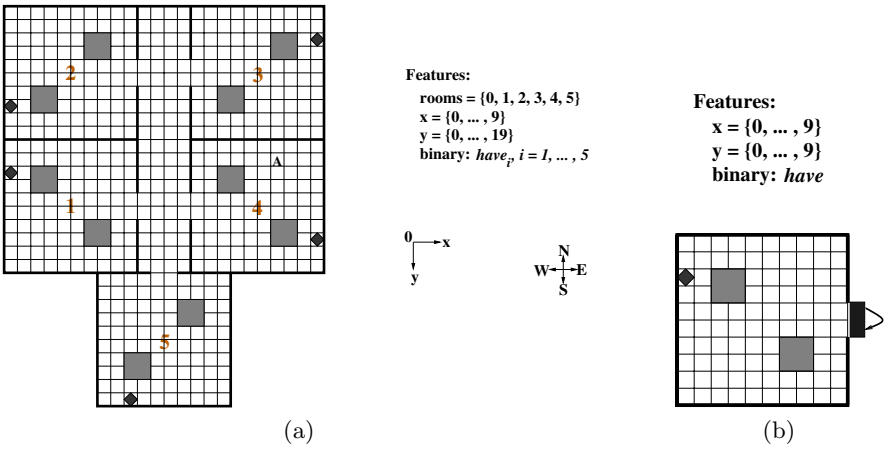



**Fig. 2.** (a) A simple rooms domain with similar rooms. The task is to collect all 5 objects in the environment. (b) The option MDP corresponding to a *get-object-and-leave-room* option. See text for full description.

the same square as the object. Each of the rooms is a 10 by 10 grid with certain obstacles in it. The actions available to the agent are $\{N, S, E, W\}$ with a 0.1 probability of *failing*, i.e., going randomly in a direction other than the intended one. The state is described by the following features: the room number the agent is in, with 0 denoting the corridor, the $x$ and $y$ co-ordinates within the room or corridor with respect to the reference direction indicated in the figure and boolean variables $have_i$, $i = 0, 1, \ldots, 5$, indicating possession of object in room $i$. Thus the state with the agent in the cell marked $A$ in the figure and having already gathered the objects in rooms 2 and 4 is represented by $\langle 3, 6, 8, 0, 1, 0, 1, 0 \rangle$. The goal is any state of the form $\langle \cdot, \cdot, \cdot, 1, 1, 1, 1, 1 \rangle$ and the agent receives a reward of $+1$ on reaching a goal state.

We compared the performance of an agent that employs relativized options with that of an agent that uses multiple regular options. The "relativized" agent employs a single relativized option whose policy can be suitably lifted to apply in each of the 5 rooms. The relativized option MDP corresponds to a single room and is shown in Figure 2(b). The state space $S'$ of the option MDP is defined by 3 features: $x$ and $y$ co-ordinates and a binary feature *have*, which is true if the agent has gathered the object in the room. There is an additional absorbing state-action pair $(\tau, \alpha)$, otherwise the action set remains the same. The stopping criterion $\beta$ is 1 at $\tau$ and zero elsewhere. The initiation set consists of all states of the form $\langle i, * \rangle$, with $i \neq 0$. There is a reward of $+1$ on transiting to $\tau$ from any state of the form $\langle *, 1 \rangle$, i.e. on exiting the room with the object.

One can see that lifting a policy defined in the option MDP yields different policy fragments depending on the room in which the option is invoked. For example, a policy in the option MDP that picks $E$ in all states would lift to

yield a policy fragment that picks $W$ in rooms 3 and 4, picks $N$ in room 5 and picks $E$ in rooms 1 and 2.

The "regular" agent employs 5 regular options, $O_1, \cdots, O_5$, one for each room. Each of the option employs the same state space and stopping criterion as the relativized option. The initiation set for option $O_i$ consists of states of the form $\langle i, * \rangle$. There is a reward of $+1$ on exiting the room with the object. Both agents employ SMDP Q-learning [3] at the higher level and Q-learning [18] at the option level.

We also compared the performance of an agent that employs only the four primitive actions. All the agents used a discount rate of 0.9, learning rate of 0.05 and $\epsilon$-greedy exploration, with an $\epsilon$ of 0.1. The results shown are averaged over 100 independent runs. The trials were terminated either on completion of the task or after 3000 steps.

Figure 3(a) shows the asymptotic performance of the agents. This graph demonstrates that the option agents perform similarly in the long run, with no significant difference in performance. The agent that employs only primitive actions takes a long time to start learning and was still improving after 50,000 steps. Since we are more interested in the initial performance of the option agents, we do not present further results for the primitive action agent.

Figure 3(b) shows the initial performance of the option agents. As expected, the relativized agent significantly outperforms the regular agent in the early trials[2]. Figure 3(c) graphs the rate at which the agents improved over their initial performance. The relativized agent achieved similar levels of improvement in performance significantly earlier than the regular option. For example, the relativized agent achieved a 60% improvement in initial performance in 40 trials, while the regular agent needed 110 trials. These results demonstrate that employing relativized options significantly speeds up initial learning performance, and if the homomorphism conditions hold exactly, there is no loss in the asymptotic performance.

## 5 Approximate Homomorphisms

The various rooms in the test bed above map exactly onto the option MDP in Figure 2(b). In practice such exact equivalences do not arise often. To study the usefulness of relativized options in inexact settings, we conducted further experiments in which the rooms had different dynamics. In the first task, the rooms had the same set of obstacles, but had different probabilities of action success. In the corridor actions fail with probability 0.1 and in rooms 1 through 5 with probabilities 0.2, 0.3, 0.25, 0.5 and 0.0 respectively. Figure 4(b) shows the initial performance of the relativized agent and the regular agent on this task. Again the relativized agent significantly outperforms the regular agent initially and the asymptotic performance, Figure 4(a), shows no significant difference.

In the second task, the rooms have differently shaped obstacles, as shown in Figure 5(a). Again there is a significant improvement in initial performance, but

[2] All the significance tests were *two sample t-tests* with a p-value of 0.01.

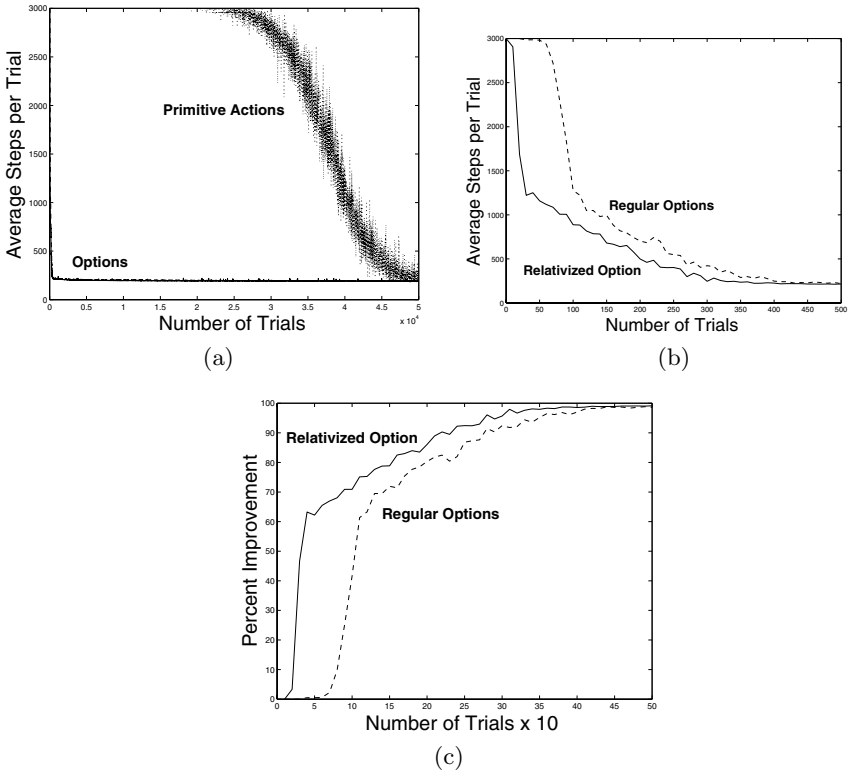



**Fig. 3.** (a) Comparison of asymptotic performance of various learning agents on the task shown in Figure 2. See text for description of the agents. (b) Comparison of initial performance of the regular and relativized agents on the same task. (c) Comparison of the rate of improvement to final performance of the two agents.

the asymptotic performance of the relativized agent is slightly, but significantly, worse than the regular agent, as shown in Figures 6(a) and 6(b). This loss in asymptotic performance is expected and is observed in other inexact scenarios we tested the agents on. In some cases this loss reaches unacceptable levels, with the relativized agent failing to successfully complete the task on certain trials even after considerable training.

One way to bound this loss in asymptotic performance is to model the option homomorphism as a map from an MDP to a *Bounded-parameter MDP* (BMDP) [7]. A BMDP is an MDP in which the transition probabilities and the rewards are specified as intervals. Formally a BMDP $\mathcal{M}'$ is given by the tuple $\langle S, A, \Psi, P_{\updownarrow}, R_{\updownarrow} \rangle$ where $S$ and $A$ are the state and action sets, $\Psi$ is the set of admissible state-action pairs, $P_{\updownarrow} : \Psi \times S \to [0,1] \times [0,1]$ with $P_{\updownarrow}(s,a,s') = [P_{low}(s,a,s'), P_{high}(s,a,s')]$, for all $(s,a)$ in $\Psi$ and $s'$ in $S$, is the range of values for the probability of transiting from $s$ to $s'$ under action $a$ and $R_{\updownarrow} : \Psi \to \mathbb{R} \times \mathbb{R}$, with $R_{\updownarrow}(s,a) =$

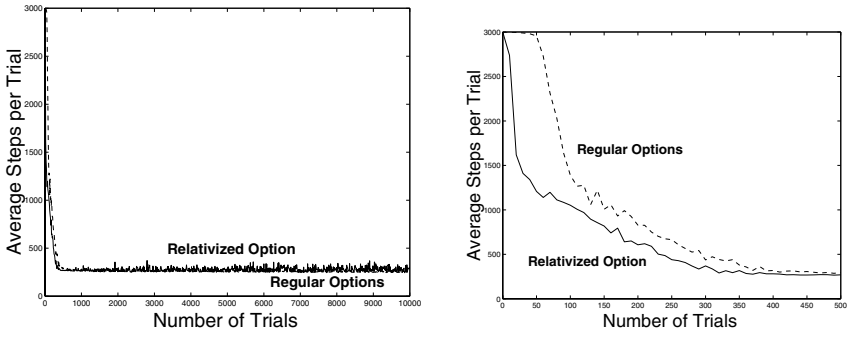



**Fig. 4.** (a) Comparison of asymptotic performance of the regular and relativized agents on the modified rooms task. See text for description of the task. (b) Comparison of initial performance of the two agents on the same task.

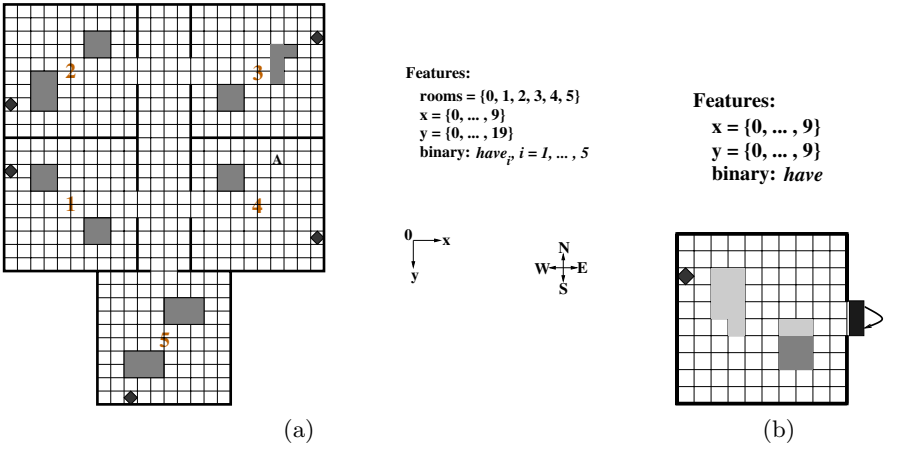



**Fig. 5.** (a) A simple rooms domain with dissimilar rooms. The task is to collect all 5 objects in the environment. (b) The option BMDP corresponding to a *get-object-and-leave-room* option. See text for full description.

$[R_{low}(s,a), R_{high}(s,a)]$, for all $(s,a)$ in $\Psi$, is the range of the expected reward on performing action $a$ in state $s$.

**Definition:** An *approximate MDP homomorphism* $h$ from an MDP $\mathcal{M} = \langle S, A, \Psi, P, R\rangle$ to a BMDP $\mathcal{M}' = \langle S', A', \Psi', P'_{\updownarrow}, R'_{\updownarrow}\rangle$ is a surjection from $\Psi$ to $\Psi'$, defined by a tuple of surjections $\langle f, \{g_s | s \in S\}\rangle$, with $h((s,a)) = (f(s), g_s(a))$, where $f : S \to S'$ and $g_s : A_s \to A'_{f(s)}$ for $s \in S$, such that, $\forall s, s' \in S$ and

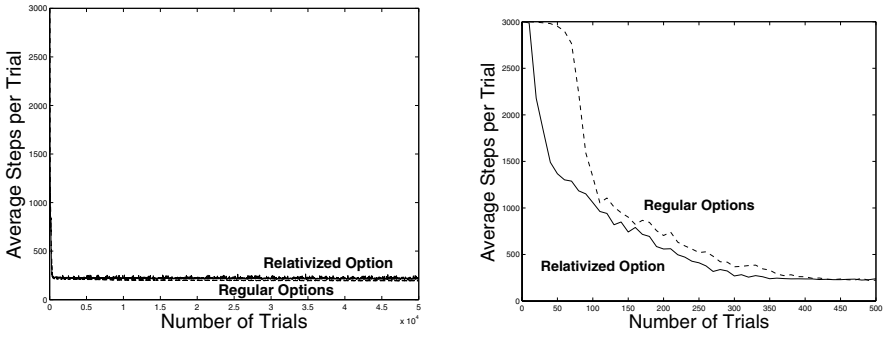



**Fig. 6.** (a) Comparison of asymptotic performance of the regular and relativized agents on the task in Figure 5. (b) Comparison of initial performance of the two agents on the same task.

$a \in A_s$:

$$P'_{\updownarrow}(f(s), g_s(a), f(s')) = \left[ \min_{t \in [s]_{B_h|S}} T(t, a, [s']_{B_h|S}), \max_{t \in [s]_{B_h|S}} T(t, a, [s']_{B_h|S}) \right] \quad (6)$$

$$R'_{\updownarrow}(f(s), g_s(a)) = \left[ \min_{t \in [s]_{B_h|S}} R(t, a), \max_{t \in [s]_{B_h|S}} R(t, a) \right] \quad (7)$$

In the rooms task depicted in Figure 5(a) the homomorphism corresponding to the relativized option may now be viewed as a map from each of the rooms to the image BMDP in Figure 5(b), where the probabilities of transiting into and out of the lightly colored states range from 0 to 1. We can now use the *interval value iteration* algorithm of Givan, Leach and Dean [7] to arrive at bounds for the optimal value function in this BMDP and hence can bound the loss of performance that arises due to employing such approximate homomorphisms.

## 6 Discussion

Our work derives mainly from the model-minimization framework of Dean and Givan [4,6]. Their work is based on concepts from FSA minimization [9] and concurrent process model checking [13]. They build their framework on the notion of stochastic bisimulations [12] from model checking and extend the definition to MDPs by incorporating the possibility of decision making and stochasticity. But they do not address the problem of state-action equivalence and symmetries. We base our work on the concept of stochastic homomorphisms derived from the FSA literature, which we believe is a simpler notion than bisimulations and helps to better understand the minimization process. Many of the results we

present in this paper are extensions of similar results obtained by Givan, Dean and Greig [6] and have counterparts in FSA minimization frameworks.

Minimization algorithms for other modeling paradigms often employ symmetry groups. For example, Jump [11] uses symmetry groups of FSA to decompose a machine into identical components, Emerson and Sistla [5] use symmetry groups to simplify models of concurrent systems, and Glover [8] employs symmetry groups in deriving shift invariant models of Markov processes.

Defining MDP symmetry groups with automorphisms on the states does not capture all the interesting cases of symmetry. Hence employing symmetry groups in Dean and Givan's framework does not give us much leverage. Extending the notion of automorphisms to state-action pairs enables us to overcome this deficiency and employ concepts from group theory and traditional minimization approaches to greater effect. To the best of our knowledge, ours is the first work to employ extended stochastic homomorphisms and symmetry groups in minimization of MDPs.

The state-of-the-art MDP minimization algorithms [1,2,6] can automatically construct reduced models of an MDP given the complete system model, i.e., a complete specification of all the components of the MDP. These algorithms can be extended to incorporate state-action equivalence and to compute reduced models as defined in Section 3. Symmetries of MDPs often have special forms. For example Zinkevich and Balch [19] explore symmetries in multi agent systems that arise from permutation of the features corresponding to various agents. Concurrent process literature [5] also abounds with examples of systems with permutation symmetry groups. Often this special form of the symmetry group leads to more efficient minimization algorithms, and we are presently investigating minimization methods that take advantage of symmetries.

Most minimization algorithms, for MDPs and other formalisms, require that we specify the complete system model. Algorithms that exploit symmetries (e.g. ref. [5]) require that we specify the symmetry group beforehand. This requires the designer to provide considerable domain knowledge to the agent, which might not be available or difficult to obtain in many cases. We are currently investigating minimization algorithms that can work with partial specification of the system model and symmetry groups and still derive reasonable reduced models of the system.

Relativized options, per se, do not necessarily add more expressive power to the options framework. It is possible to achieve the same decomposition of a problem by employing regular options. But if we are learning the option policy online, then we garner considerable advantage in terms of efficiency and speed. When employing a relativized option we can employ the experience generated by every invocation of the option to learn a policy on a much smaller image MDP. Thus relativized options allow us to considerably speed up learning and make more efficient use of online experience. They also give us the power to specify a single option that can be applied to many symmetrically equivalent situations, as in the task in Figure 2, where the various rooms are symmetrically equivalent.

In this work we demonstrated that the predicted speed up when employing relativized options is achieved in practice. We also demonstrated that relativized options are useful even in cases where the homomorphism conditions are not satisfied exactly. We employ Bounded-parameter MDPs [7] to characterize approximate homomorphisms and to bound the loss in performance when the homomorphism conditions are not met exactly. The bound on the loss can also be used in establishing conditions under which an option might be relativized and to guide the search for a suitable homomorphism. Our current research is focussed on defining principled ways to generate relativized options.

## 7 Conclusion

In this paper we presented an MDP minimization framework based on the notion of MDP homomorphism. This is an extension of Dean and Givan's model minimization framework. Our framework can accommodate state-action equivalence and explicitly addresses the issue of modeling symmetries of MDPs. We then developed the concept of partial homomorphisms and applied our minimization framework to hierarchical reinforcement learning to define relativized options—compact, symmetry invariant options. We empirically demonstrated the usefulness of relativized options even in case the homomorphism conditions are not met exactly. We developed the notion of approximate homomorphisms that allow us to bound the loss of performance in such cases.

**Acknowledgements.** We wish to thank Dan Bernstein, Amy McGovern and Mike Rosenstein for many hours of useful discussion. This material is based upon work supported by the National Science Foundation under Grant No. ECS-9980062 to Andrew G. Barto. 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. C. Boutilier and R. Dearden. Using abstractions for decision theoretic planning with time constraints. In *Proceedings of the AAAI-94*, pages 1016–1022. AAAI, 1994.
2. C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. In *Proceedings of International Joint Conference on Artificial Intelligence 14*, pages 1104–1111, 1995.
3. Steven J. Bradtke and Michael O. Duff. Reinforcement learning methods for continuous-time Markov decision problems. In *Advances in Neural Information Processing Systems 7*. MIT Press, 1995.
4. Thomas Dean and Robert Givan. Model minimization in markov decision processes. In *Proceedings of AAAI-97*, pages 106–111. AAAI, 1997.
5. E. A. Emerson and A. P. Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1/2):105–131, 1996.

6. Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimization in markov decision processes. Submitted to Artificial Intelligence, 2001.
7. Robert Givan, Sonia Leach, and Thomas Dean. Bounded-parameter markov decision processes. *Artificial Intelligence*, 122:71–109, 2000.
8. J. Glover. Symmetry groups and translation invariant representations of markov processes. *The Annals of Probability*, 19(2):562–586, 1991.
9. J. Hartmanis and R. E. Stearns. *Algebraic Structure Theory of Sequential Machines*. Prentice-Hall, Englewood Cliffs, NJ, 1966.
10. Glenn A. Iba. A heuristic approach to the discovery of macro-operators. *Machine Learning*, 3:285–317, 1989.
11. J. R. Jump. A note on the iterative decomposition of finite automata. *Information and Control*, 15:424–435, 1969.
12. K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.
13. D. Lee and M. Yannakakis. Online minimization of transition systems. In *Proceedings of* $24^{\text{th}}$ *Annual ACM Symposium on the Theory of Computing*, pages 264–274. ACM, 1992.
14. Doina Precup. *Temporal Abstraction in Reinforcement Learning*. PhD thesis, University of Massachusetts, Amherst, May 2000.
15. B. Ravindran and A. G. Barto. Symmetries and model minimization of markov decision processes. Technical Report 01-43, University of Massachusetts, Amherst, 2001.
16. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning. An Introduction*. MIT Press, Cambridge, MA, 1998.
17. Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and Semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.
18. C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, Cambridge University, Cambridge, England, 1989.
19. M. Zinkevich and T. Balch. Symmetry in markov decision processes and its implications for single agent and multi agent learning. In *Proceedings of the 18th International Conference on Machine Learning*, pages 632–640, San Francisco, CA, 2001. Morgan Kaufmann.

# Learning Options in Reinforcement Learning

Martin Stolle and Doina Precup

School of Computer Science,
McGill University,
Montreal, QC, H3A 2A7, Canada
{mstoll,dprecup}@cs.mcgill.ca,
http://www.cs.mcgill.ca/{~mstoll,~dprecup}

**Abstract.** Temporally extended actions (e.g., macro actions) have proven very useful for speeding up learning, ensuring robustness and building prior knowledge into AI systems. The options framework (Precup, 2000; Sutton, Precup & Singh, 1999) provides a natural way of incorporating such actions into reinforcement learning systems, but leaves open the issue of how good options might be identified. In this paper, we empirically explore a simple approach to creating options. The underlying assumption is that the agent will be asked to perform different goal-achievement tasks in an environment that is otherwise the same over time. Our approach is based on the intuition that states that are frequently visited on system trajectories, could prove to be useful subgoals (e.g., McGovern & Barto, 2001; Iba, 1989).
We propose a greedy algorithm for identifying subgoals based on state visitation counts. We present empirical studies of this approach in two gridworld navigation tasks. One of the environments we explored contains bottleneck states, and the algorithm indeed finds these states, as expected. The second environment is an empty gridworld with no obstacles. Although the environment does not contain any obvious subgoals, our approach still finds useful options, which essentially allow the agent to explore the environment more quickly.

## 1 Introduction

Temporally extended actions (e.g., macro actions) have proven very useful for speeding up learning and planning, ensuring robustness and building prior knowledge into AI systems (e.g., Fikes, Hart, and Nilsson, 1972; Newell and Simon, 1972; Sacerdoti, 1977; Korf, 1985; Laird et al., 1986; Minton, 1988; Iba, 1989). More recently, the topic has been explored in the context of Markov Decision Processes (MDPs) and reinforcement learning (RL) (e.g., Singh, 1992; Parr, 1998; Dietterich, 1998; Precup, 2000). The options framework (Precup, 2000; Sutton, Precup & Singh, 1999) provides a natural way of incorporating extended actions into reinforcement learning systems. An *option* is specified by a set of states in which the option can be initiated, an internal policy and a termination condition. One natural way to define the termination condition for an option is to choose a state (or set of states) as "subgoals" that the option should achieve (see, e.g., Precup, 2000). The option terminates when the system enters a subgoal state. Termination can also be forced on "failure conditions" (e.g., if the option has been executing for too long). If the initiation set and the termination condition are specified,

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 212–223, 2002.

traditional reinforcement learning methods can be used to learn the internal policy of the option.

While the options framework provides a convenient language for describing and reasoning with temporally extended actions, it gives no guidance regarding how good options can be found. The issue of automatically finding initiation sets and termination conditions for options is left open. In this paper, we propose and explore empirically a simple approach for creating options. We consider a scenario in which an agent will be asked to perform different goal-achievement tasks in an environment that is otherwise the same over time. Our approach is based on the intuition that states that are frequently visited on system trajectories could prove to be useful subgoals. This idea was explored by Iba (1989) as a heuristic for discovering macro-operators in deterministic search problems. Recently, similar ideas were used by McGovern (2002; McGovern & Barto, 2001) in the context of discovering options. The main idea of their approach is to detect "bottleneck" regions in the state space, by using the concept of diverse density. Bottleneck states are defined as states the occur often on "good" trajectories and do not occur on "bad" trajectories. Diverse density is a natural computational tool for identifying such states.

Our approach is similar in spirit to McGovern's, but more simple, and based on different assumptions. First, we do not assume a priori that there may be "good" and "bad" trajectories. Instead, we assume that the agent will be confronted with different tasks, all set in the same environment. Hence, the agent is first allowed to explore the environment and gather information. We assume that frequently visited states are of potential interest, and should become subgoals (regardless of the quality of the trajectories on which they occur). Second, our approach explicitly constructs a *set of options* rather than individual options. McGovern focuses on constructing individual options one at a time, and then uses additional filters in order to construct the option set.

At the beginning of learning, our agent is allowed to interact with the environment and gather statistics, focusing on state visitation counts. Based on these statistics, it chooses potential subgoals and initiation states for the options. In the second phase, the agent learns the internal policies of the options, using standard reinforcement learning techniques. Once these are learned, the agent can use the options to solve the goal-achievement tasks with which it is presented.

We present empirical studies of this approach in two gridworld navigation tasks. One of the environments we explored contains bottleneck states, and the algorithm indeed finds these states, as expected. If the algorithm is allowed to gather a sufficient amount of experience, it discovers useful options, which accelerate learning on future tasks. The performance of the options discovered automatically is very similar to that of carefully hand-crafted options.

The second environment is an empty gridworld with no obstacles. Although the environment does not contain any obvious subgoals that a human designer can exploit, our approach still finds useful options, which essentially allow the agent to explore the environment more quickly.

The paper is organized as follows. In Section 2 we introduce basic reinforcement learning notation. Section 3 contains the definition of options, SMDP Q-learning, and intra-option Q-learning, the two main algorithms we use for learning how to choose

among options. In Section 4 we describe our algorithm for automatically creating options. Section 5 contains empirical results for this algorithm on two gridworld tasks, and a discussion of the behavior we observed. In section 6 we outline some ideas for future work.

## 2 Reinforcement Learning

Reinforcement learning (RL) is a computational approach to automating goal-directed learning and decision making (Sutton & Barto, 1998). It encompasses a broad range of methods for determining optimal ways of behaving in complex, uncertain and stochastic environments. Most current RL research is based on the theoretical framework of Markov Decision Processes (MDPs) (Puterman, 1996). MDPs are a standard, very general formalism for studying stochastic, sequential decision problems. In this framework, the agent takes a sequence of primitive actions paced by a discrete, fixed time scale. On each time step $t$, the agent observes the state of its environment, $s_t$, contained in a finite discrete set $\mathcal{S}$, and chooses an action, $a_t$, from a finite action set $\mathcal{A}$ (possibly dependent on $s_t$). One time step later, the agent receives a reward $r_{t+1}$ and the environment transitions to a next state, $s_{t+1}$. For a given state $s$ and action $a$, the expected value of the immediate reward is $r_s^a$ and the transition to a new state $s'$ has probability $p_{ss'}^a$, regardless of the path taken by the agent before state $s$. A way of behaving, or *policy*, is defined as probability distribution for picking actions in each state: $\pi : \mathcal{S} \times \mathcal{A} \to [0,1]$. The goal of the agent is to find a policy that maximizes the total reward received over time. For any policy $\pi$ and any state $s \in \mathcal{S}$, the value of taking action $a$ in state $s$ under policy $\pi$, denoted $Q^\pi(s,a)$, is the expected discounted future reward starting in $s$, taking $a$, and henceforth following $\pi$:

$$Q^\pi(s,a) \stackrel{\text{def}}{=} E_\pi\Big\{ r_{t+1} + \gamma r_{t+2} + \cdots \;\Big|\; s_t = s, a_t = a \Big\}.$$

The *optimal* action-value function is:

$$Q^*(s,a) \stackrel{\text{def}}{=} \max_\pi Q^\pi(s,a).$$

In an MDP, there exists a unique optimal value function, $Q^*(s,a)$, and at least one optimal policy, $\pi^*$, corresponding to this value function:

$$\pi^*(s,a) > 0 \text{ iff } a \in \arg\max_{a' \in \mathcal{A}} Q^*(s,a').$$

Many popular reinforcement learning algorithms aim to compute $Q^*$ (and thus implicitly $\pi^*$) based on the observed interaction between the agent and the environment. The most widely-used is probably Q-learning (Watkins, 1989), which allows learning $Q^*$ directly from interaction with the environment. The agent updates its value function as follows:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha(r_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q(s_{t+1},a') - Q(s_t,a_t))$$

Q-learning has been shown to converge in the limit, with probability 1, to the optimal value function $Q^*$, under standard stochastic approximation assumptions.

## 3 Options

Options (Precup, 2000; Sutton, Precup & Singh, 1999) are a generalization of primitive actions to include temporally extended courses of action. Options consist of three components: a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0,1]$, a termination condition $\beta : \mathcal{S} \to [0,1]$, and an initiation set $\mathcal{I} \subseteq \mathcal{S}$. An option $< \mathcal{I}, \pi, \beta >$ is available in state $s$ if and only if $s \in \mathcal{I}$. If the option is taken, then actions are selected according to $\pi$ until the option terminates stochastically according to $\beta$. That is, in state $s_t$ the next action $a_t$ is selected according to the probability distribution $\pi(s_t, \cdot)$. The environment then makes a transition to state $s_{t+1}$, where the option either terminates, with probability $\beta(s_{t+1})$, or else continues, determining $a_{t+1}$ according to $\pi(s_{t+1}, \cdot)$, possibly terminating in $s_{t+2}$ according to $\beta(s_{t+2})$, and so on. When the option terminates, then the agent has the opportunity to select another option. In this paper we focus mainly on options which have a deterministic internal policy $\pi$. We use the notation $\pi(s)$ to denote the primitive action chosen by policy $\pi$ in state $s$.

Note that primitive actions can be viewed as a special case of options. Each action $a$ corresponds to an option that is available whenever $a$ is available ($\mathcal{I} = \{s : a \in \mathcal{A}_s\}$), that always lasts exactly one step ($\beta(s) = 1, \ \forall s \in \mathcal{S}$), and that selects $a$ everywhere ($\pi(s) = a, \ \forall s \in \mathcal{I}$). Thus, we can consider the agent's choice at each time to be entirely among options, some of which persist for a single time step, others which are temporally extended.

It is natural to generalize the action-value function to an *option*-value function. We define $Q^\mu(s,o)$, the value of taking option $o$ in state $s \in \mathcal{I}$ under policy $\mu$, as

$$Q^\mu(s,o) \doteq E\left\{ r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \;\middle|\; \mathcal{E}(o\mu, s, t) \right\}, \tag{1}$$

where $o\mu$ denotes the policy that first follows $o$ until it terminates and then initiates $\mu$ in the resulting state, and $\mathcal{E}(o\mu, s, t)$ denotes the execution of $o\mu$ in state $s$ starting at time $t$.

Options are closely related to the actions in a special kind of decision problem known as a *semi-Markov decision process*, or *SMDP* (e.g., see Puterman, 1994). In fact, any MDP with a fixed set of options is an SMDP. Accordingly, the theory of SMDPs provides an important basis for a theory of options.

The problem of finding an optimal policy over a set of options $\mathcal{O}$ can be addressed by learning methods. Because the MDP augmented by the options is an SMDP, we can apply SMDP learning methods as developed by Bradtke and Duff (1995), Parr and Russell (1997), Mahadevan (1996), or McGovern, Sutton and Fagg (1997). When the execution of option $o$ is started in state $s$, we next jump to the state $s'$ in which $o$ terminates. Based on this experience, an option-value function $Q(s,o)$ is updated. For example, the SMDP version of one-step Q-learning (Bradtke and Duff, 1995), updates the value function after each option termination by

$$Q(s,o) \leftarrow Q(s,o) + \alpha \left[ r + \gamma^k \max_{a \in \mathcal{O}} Q(s',a) - Q(s,o) \right], \tag{2}$$

where $k$ denotes the number of time steps elapsing between $s$ and $s'$, $r$ denotes the cumulative discounted reward over this time, and it is implicit that the step-size parameter

$\alpha$ may depend arbitrarily on the states, option, and time steps. The estimate $Q(s,o)$ converges to the optimal value function over options, $Q^*_{\mathcal{O}}(s,o)$ for all $s \in \mathcal{S}$ and $o \in \mathcal{O}$ under conditions similar to those for conventional Q-learning (Parr, 1998).

One drawback of SMDP learning methods is that they need to execute an option to termination before learning can take place. Therefore, they are hard to use in the case in which options take a long time to execute, and can only be applied to one option at a time. An alternative to this approach is to use *intra-option learning* methods (Sutton, Precup & Singh, 1998), which permit learning about an option from the experience at every time step. In this paper, we use the intra-option Q-learning algorithm. On every time step $t$, after the primitive action $a_t$ is chosen, we determine all the options $o = \langle I, \pi, \beta \rangle$ for which $\pi(s_t) = a_t$. We update the values of all these options using the following update rule:

$$Q(s_t,o) \leftarrow Q(s_t,o) + \alpha(r_{t+1} + \gamma U(s_{t+1},o) - Q(s_t,o)) \tag{3}$$

where

$$U(s,o) = (1-\beta(s))Q(s,o) + \beta(s)\max_{o'} Q(s,o').$$

This algorithm converges to the correct optimal option-value function, under standard stochastic approximation conditions (Precup, 2002). Intra-option learning is potentially more efficient that SMDP learning, because it extracts more training information from the same amount of experience in the environment. This approach is also more flexible than SMDP learning, because it allows learning the value of options that are never completely executed, as long as the primitive actions taken are consistent with those options.

## 4 Finding Options

If an initiation set and a termination condition are specified, the internal policies of options can be learned using standard RL methods (e.g., Q-learning, like in Precup, 2000). But the key issue remains how to select good termination conditions and initiation sets. Intuitively, it is clear that different heuristics would work well for different kinds of environments.

In this paper, we assume that the agent is confronted with goal-achievement tasks, which take place in a fixed environment. This setup is quite natural when thinking about human activities. For instance, every day we might be cooking a different breakfast, but the kitchen layout is the same from day to day.

We allow the agent to explore the environment ahead of time, in order to learn options. The option finding process is based on posing a series of random tasks in this environment and letting the agent solve them. During this time, the agent gathers statistics regarding the frequency of occurrence of different states. Our algorithm is based on the intuition that, if states occur frequently on trajectories that represent solutions to random tasks, then these states may be important. Hence, we will learn options that use these states as targets. The options finding algorithm is described in detail below.

1. Select a number of start states $S$ and target states $T$, according to a given distribution. These states will be used to generate random tasks for the agent. In the absence of

domain knowledge, a natural assumption is to consider a uniform distribution over start and target states. However, if we have some knowledge about the types of task the agent may be facing, that knowledge can be built into the choice of distribution.

2. For each pair $\langle S, T \rangle$
   a) Perform $N_{train}$ episodes of Q-learning, to learn a policy for going from $S$ to $T$
   b) Perform $N_{test}$ episodes using the greedy policy learned. For all states $s$, count the total number of times $n(s)$ that each state is visited during these trajectories.
3. Repeat until the desired number of options is reached:
   a) Pick the state with the most visitations, $T_{max} = \arg\max_s n(s)$, as the target state for the option
   b) Compute $n(s, T_{max})$, the number of times each state $s$ occurs on paths that go through $T_{max}$
   c) Compute $\theta(T_{max}) = avg_s n(s, T_{max})$.
   d) Select all the states $s$ for which $n(s, T_{max}) > \theta(T_{max})$ to be part of the initiation set $I$ for the option.
   This step selects for the initiation set states that more likely to be on trajectories going to $T_{max}$. This is a heuristic choice, based on the intuition that $T_{max}$ is more easily reachable from these states. Hence, an option starting in these states would have a good chance of achieving its subgoal (perhaps even in a short amount of time).
   e) Complete the initiation set by interpolating between the selected states. The interpolation process is domain-specific.
   f) Decrease the visitation counts for all states by the number of visits to the states on trajectories going to $T_{max}$. The goal of this step is to prevent several options in the set we are creating to go to "neighboring" subgoals.
4. For each option, learn its internal policy; this is achieved by giving a high reward for entering $T_{max}$, and no rewards otherwise. The agent performs Q-learning, by performing episodes which start at random states in $\mathcal{I}$ and end when $T_{max}$ is reached, or when the agent exits $\mathcal{I}$.

Once the options are found, we can use either SMDP Q-learning, or intra-option Q-learning in order to learn a policy over options (as described in the previous section).

## 5 Experimental Results

We experimented with this algorithm for finding options in two simple gridworld navigation tasks. The first one is the rooms environment depicted in Figure 1 (left panel). The state is the current cell position. There are four stochastic primitive actions which move the agent up, down, left and right. Each action moves the agent in the expected direction with probability 0.9. With probability 0.1, the agent is moved instead in one of the other three directions, chosen at random. If the agent attempts to move into a wall, it stays in the same position, and no penalty is incurred. The discount factor is $\gamma = 0.9$ and there are no intermediate rewards anywhere. The agent can only obtain a reward upon entering a designated goal state. We assume that the goal state may move from one location to another over time.

The hallway states are obvious candidates for option subgoals in this environment, since trajectories passing from one room to another have to pass through the hallways. Hence, one would expected the option finding algorithm to work well, finding options for going to the hallways.

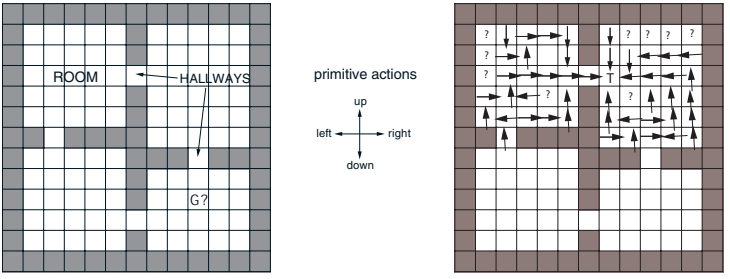



**Fig. 1.** Rooms environment, and example option

We performed 30 independent runs of the algorithm. During each run, the option finding algorithm was used to find 8 options. During the option finding stage, 25% of the states were used as potential start and goal states for random tasks. We used $N_{train} = 200$ episodes to learn Q-values for each pair of start and goal states. After learning, we used $N_{test} = 10$ episodes to generate the state visitation counts.

Recall that the algorithm selects some states to be part of the initiation set; these states will be the ones occurring frequently on trajectories that go to the subgoal state of the option. The initiation sets were then constructed by creating a rectangular box around the selected initiation states for each option. This approach is specific for gridworld navigation, and would not be expected to work well in other domains. However, other interpolation techniques may be more widely applicable. Note also that initiation sets could include just the states selected by the algorithm, without any interpolation. After the initiation sets and the targets of the options were selected, the internal policy of each option was found using Q-learning for 100 trials.

One example of an option found by the algorithm is depicted in Figure 1 (right panel). The target state for the option is close to one of the hallways, but not exactly in the hallway. This behavior was consistent across all runs, and is due to the fact that the states near the hallways are traversed both by trajectories going between rooms and by trajectories inside a room. This is also consistent with earlier findings of McGovern (1998). The figure shows that the option policy has been learned for most initiation states, although some states have a suboptimal policy, while others (marked with a '?') have still not been visited (and hence have a random policy).

Once the options were learned, we compared the performance of three learning agents: one using primitive actions only, and two using using both primitive actions and options, but with different learning algorithms (SMDP Q-learning and intra-option Q-learning respectively). All action-value functions were initialized to zero values. We individually optimized the learning rate parameter $\alpha$ for each agent, but all agents per-

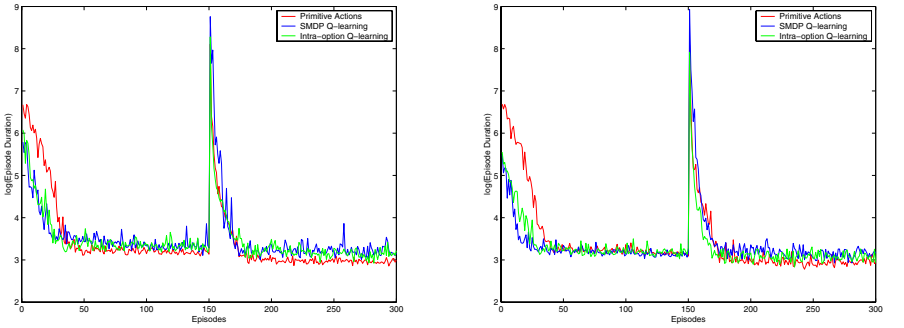



**Fig. 2.** Learning curves during training. The left panel shows the average duration of learning episodes with automatically discovered options. The right panel shows the similar plot with engineered options.

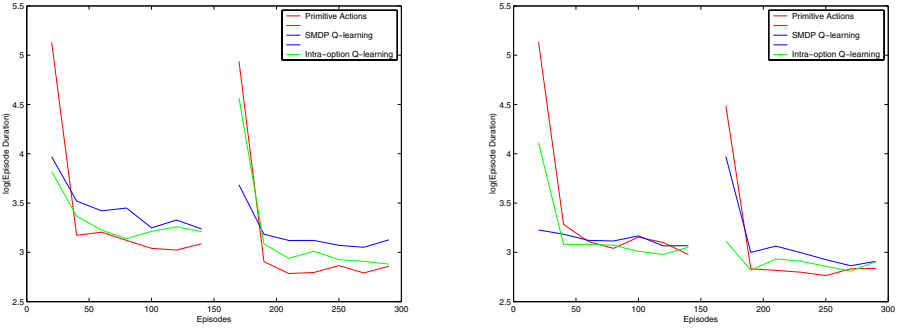



**Fig. 3.** Performance during testing episodes. The left panel shows the performance of the greedy policy over the automatically discovered options, while the right panel shows a similar plot for the engineered options.

formed best with a learning rate of $\alpha = 0.5$. During training, all agents used an $\epsilon$-greedy policy for generating behavior, with $\epsilon = 0.1$.

We picked two different goal locations. The agents were trained for 150 episodes using the first location (at coordinates (8,8), in the lower right room). Every episode consists of starting at a randomly chosen location and performing navigation actions until the goal is reached. After each 20 training episodes, we performed 5 test episodes, starting at random states and using the greedy policy to choose actions. After 150 episodes, the goal was moved to location (4,4), in the upper left room, but the Q-values of each agent were retained, and the whole process was repeated. The reason for retaining the Q-values was that we wanted to capture the situation in which the agent is not explicitly told that the task has changed, but has to discover this through trial and error. This also allows us to characterize performance expectations when options are used to learn a policy from scratch, compared to the case in which the agent's initial value function reflects previous experience.

We ran a similar set of experiments using human engineered options. In this case, we explicitly designated the hallway states as end points for the options, and we limited the initiation sets to the states within a room (like in Sutton, Precup & Singh, 1999). Then, we used Q-learning to learn internal policies for the options. We ran Q-learning sufficiently long, to allow a good policy to be learned from all initiation states. The learning algorithms, exploration policy and learning rates are the same as described above. The main goal of this experiment was to test the extent to which the "imperfections" in the options affect the speed of learning, and the performance of the learned policy.

The results for the rooms environment are depicted in Figure 2 and Figure 3. Figure 2 shows the log of the length of the learning episodes, averaged over the 30 runs. We used log-scale to improve the readability of the graphs. Figure 3 shows the same performance measure for the testing episodes. Each point on this graph represents the average of the five test episodes. In each figure, the left panel shows the performance of the automatically discovered options, while the right panel shows the performance of the human engineered options.

As expected, when learning is started with a clean slate (first half of the graphs) using temporally extended options provides an advantage over using primitive actions only, although the number of actions to learn about has increased from 4 to 12. During training, the performance of the options discovered by our algorithm was virtually the same as that of the hand-crafted options. During testing, the hand-crafted options performed slightly better, but not by much.

When we changed the goal location, all three algorithms performed similarly during training (second half of the graphs, after episode 150). During testing, results are mixed, as seen in Figure 3. We initially expected the options to perform significantly better in this situation, but this was not the case. Options improve their performance quicker in the beginning, but then seem to perform worse than the primitive actions. The main reason seems to be that we require, for all learners, that the options be executed to completion. If the initial value function makes poor choices, the presence of the options causes the agent to move further away from the goal. There are two possible ways to alleviate this problem: using more aggressive exploration, or allowing the options to terminate early (Sutton, Precup & Singh, 1999). We plan to explore this issue further in the future.

The second environment we experimented with was an empty $11 \times 11$ gridworld, with no obstacles. In this environment, there are no obvious states that should be designated as subgoals. Hence, it is not obvious that the option-finding heuristic would work well. Our initial expectation was that the use of the options found would actually slow down learning.

We applied our algorithm to this environment in the same manner described above. The results are depicted in Figure 4. During training, it is clear that the use of temporally extended options is helpful. The result is not so clear for the greedy policy, though. In order to understand better the behavior of the algorithm, we looked at the options that were found in this environment. On every run, the algorithm first found one (or a couple) of options leading to one of the central states in the environment. Once the state visitation counts were decreased by eliminating the trajectories that went through the center states, the peak counts were spread over states in different areas of the grid. Hence, on virtually every run, the algorithm found some options for navigating to different parts of the grid.

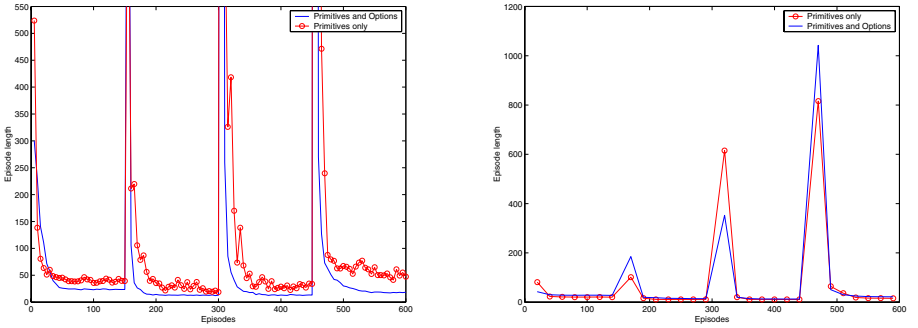



**Fig. 4.** Results for the empty gridworld environment. The left panel shows the performance during training, and the right panel shows the performance of the greedy policy learned.

Of course, such options are useful in the case in which the goal state moves around the environment. One example of option subgoal states found by our algorithm is depicted in Figure 5. The subgoals are numbered in the order in which they were selected.

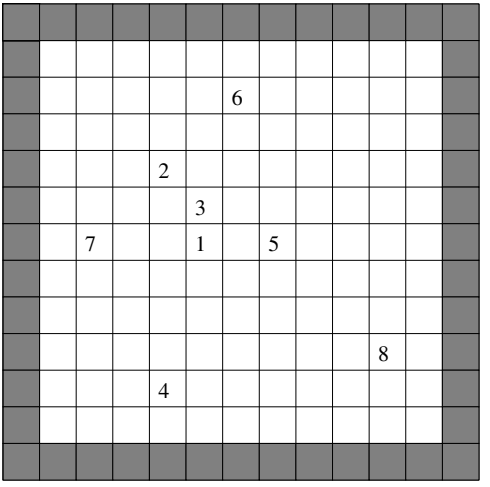



**Fig. 5.** Option target states found during one run in the empty gridworld.

## 6 Future Work

There are many possible directions for future research. Our algorithm currently relies on allowing the agent to get a fairly large amount of experience in the MDP, before it starts solving tasks. This experience could be used differently, and potentially in a more fruitful way. For instance, we could estimate a low-level model of the MDP, or a model

of the options, and then use these models to plan solutions to tasks, when they arise. We plan to compare such model-based methods to the model-free algorithm described in this paper.

Another important issue is generalizing the algorithm in large state spaces. In this paper, we assumed that we could afford to keep visitation counts for individual states, but this would not be true in very large environments. In such a case, using state aggregation, or some form of clustering, may prove fruitful.

**Acknowledgments.** This work was supported by research grants from NSERC and FCAR. The authors thank Ted Perkins and the anonymous reviewers for very useful comments on earlier drafts of this paper.

## References

[Bradtke & Duff, 1995] Bradtke and Duff][1995]Bradtke:SMDPQ Bradtke, S. J., & Duff, M. O. (1995). Reinforcement learning methods for continuous-time Markov Decision Problems. *Advances in Neural Information Processing Systems 7* (pp. 393–400). MIT Press.

[Dietterich, 1998] Dietterich][1998]Dietterich:MAXQ Dietterich, T. G. (1998). The MAXQ method for hierarchical reinforcement learning. *Proceedings of the Fifteenth International Conference on Machine Learning*. Morgan Kaufmann.

[Fikes et al., 1972] Fikes et al.][1972]Fikes:RobotPlan Fikes, R., P.E.Hart, & Nilsson, N. J. (1972). Learning and executing generalized robot plans. *Artificial Intelligence*, *3*, 251–288.

[Iba, 1989] Iba][1989]Iba:Macro Iba, G. A. (1989). A heuristic approach to the discovery of macro-operators. *Machine Learning*, *3*, 285–317.

[Korf, 1985] Korf][1985]Korf:Macro Korf, R. E. (1985). *Learning to solve problems by searching for macro-operators*. Pitman Publishing Ltd.

[Laird et al., 1986] Laird et al.][1986]Laird:ChunkSOAR Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning*, *1*, 11–46.

[Mahadevan et al., 1997] Mahadevan et al.][1997]Mahadevan:SMDP Mahadevan, S., Marchallek, N., Das, T. K., & Gosavi, A. (1997). Self-improving factory simulation using continuous-time average-reward reinforcement learning. *Proceedings of the Fourteenth International Conference on Machine Learning* (pp. 202–210). Morgan Kaufmann.

[McGovern et al., 1997] McGovern et al.][1997]McGovern:MacroRL McGovern, A., Sutton, R. S., & Fagg, A. H. (1997). Roles of macro-actions in accelerating reinforcement learning. *Grace Hopper Celebration of Women in Computing* (pp. 13–17).

[McGovern, 2002] McGovern][2002]McGovern:Thesis McGovern, E. A. (2002). *Autonomous discovery of temporal abstractions from interaction with an environment*. Doctoral dissertation, University of Massachusetts, Amherst.

[McGovern & Barto, 2001] McGovern and Barto][2001]McGovern:ICML McGovern, E. A., & Barto, A. G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. *Proceedings of the Eighteenth International Conference on Machine Learning* (pp. 361–368). Morgan Kaufman.

[Minton, 1988] Minton][1988]Minton:Book Minton, S. (1988). *Learning search control knowledge. An explanation-based approach*. Kluwer Academic Publishers.

[Newell & Simon, 1972] Newell and Simon][1972]Newell:Simon Newell, A., & Simon, H. A. (1972). *Human problem solving*. Prentice-Hall.

[Parr, 1998] Parr][1998]Parr:Thesis Parr, R. (1998). *Hierarchical control and learning for Markov Decision Processes*. Doctoral dissertation, Computer Science Division, University of California, Berkeley, USA.

[Parr & Russell, 1998] Parr and Russell][1998]Parr:HAMs Parr, R., & Russell, S. (1998). Reinforcement learning with hierarchies of machines. *Advances in Neural Information Processing Systems 10*. MIT Press.

[Precup, 2000] Precup][2000]Precup:Thesis Precup, D. (2000). *Temporal abstraction in reinforcement learning*. Doctoral dissertation, Department of Computer Science, University of Massachusetts, Amherst, USA.

[Puterman, 1994] Puterman][1994]Puterman:Book Puterman, M. L. (1994). *Markov Decision Processes: Discrete stochastic dynamic programming*. Wiley.

[Sacerdoti, 1974] Sacerdoti][1974]Sacerdoti:PlanArt Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, *5*, 115–135.

[Singh, 1992] Singh][1992]Singh:HDynaAAAI Singh, S. P. (1992). Reinforcement learning with a hierarchy of abstract models. *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 202–207). MIT/AAAI Press.

[Sutton & Barto, 1998] Sutton and Barto][1998]Sutton:Book Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT Press.

[Sutton et al., 1999] Sutton et al.][1999]Precup:Options Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*, 181–211.

[Watkins, 1989] Watkins][1989]Watkins:Qlearn Watkins, C. J. C. H. (1989). *Learning with delayed rewards*. Doctoral dissertation, Psychology Department, Cambridge University, Cambridge, UK.

# Approximation Techniques for Non-linear Problems with Continuum of Solutions

Xuan-Ha Vu, Djamila Sam-Haroud, and Marius-Calin Silaghi

Artificial Intelligence Laboratory, Institute of Core Computing Science,
School of Computer and Communication Sciences, Swiss Federal Institute of Technology,
CH-1015, Lausanne, Switzerland
{xuan-ha.vu, jamila.sam, marius.silaghi}@epfl.ch
http://liawww.epfl.ch

**Abstract.** Most of the working solvers for numerical constraint satisfaction problems (NCSPs) are designed to delivering *point-wise* solutions with an arbitrary accuracy. When there is a *continuum of feasible points* this might lead to prohibitively verbose representations of the output. In many practical applications, such large sets of solutions express equally relevant alternatives which need to be identified as completely as possible. The goal of this paper is to show that by using appropriate approximation techniques, explicit representations of the solution sets, preserving both accuracy and completeness, can still be proposed for NCSPs with continuum of solutions. We present a technique for constructing *concise* inner and outer approximations as unions of interval boxes. The proposed technique combines a *new splitting strategy* with the *extreme vertex representation* of orthogonal polyhedra [1,2,3], as defined in computational geometry. This allows for compacting the representation of the approximations and improves efficiency.

## 1 Introduction

Numerical constraints can naturally model a wide range of real-world problems. In practice, process descriptions, cost restrictions, chemical or mechanical models are most often expressed using this type of constraints. A numerical constraint satisfaction problem (NCSP), $(\mathcal{V}, \mathcal{C}, \mathcal{D})$, is stated as a set of variables $\mathcal{V}$ taking their values in domains $\mathcal{D}$ over the reals and subject to constraints $\mathcal{C}$. The constraints can be equalities or inequalities of arbitrary type and arity, usually expressed using arithmetic expressions. The goal is to assign values to the variables so that all the constraints are satisfied. Such an assignment is then called a solution. The completeness of a solving procedure means its ability to find a solution to the NCSP if any, or else, to prove that there are no solutions to the problem. Completeness is essential in many real-world situations since it is the only way to guarantee that all inconsistencies are avoided, that all relevant alternatives can be provided and that an eventual global optima can be identified. When devising complete solving techniques for NCSPs, a fundamental issue is the representation of the actual solution sets. The spectrum of possible representations ranges from the implicit one, given by the arithmetic expressions of constraints, to the explicit one, given by the enumeration of all individual solutions. The former representation is traditionally used by mathematical programming solvers. It is compact but difficult to query. As a consequence, even

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 224–241, 2002.

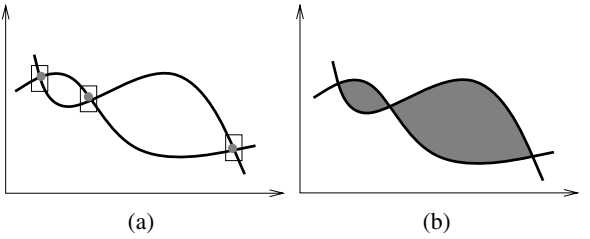



**Fig. 1.** (a) Numerical CSP with three point-wise solutions (grey dots); (b) Numerical CSP with a continuum of solutions (grey regions)

the state-of-the-art solving techniques cannot guarantee completeness in the general case [4]. The second representation is usually constructed by constraint programming solvers [5,6]. It is complete and trivial to query but can be exceedingly verbose, and therefore unpractical, when the NCSP has a continuum of solutions (see Figure 1). Such a situation often occurs in real-world applications since under-constraint problems or problems with inequalities are ubiquitous in practice. The goal of this paper is to show that by using appropriate approximation techniques, explicit representations, preserving both accuracy and completeness, can still be proposed for NCSPs with non-isolated solutions. We propose to use the *Extreme Vertex Representation* (EVR) of orthogonal polyhedra [1,2,3] as defined in computational geometry, coupled with adapted branching strategies, to compute inner and outer approximations of the solution sets under the form of *unions of interval boxes*. The resulting technique applies to general constraint systems. The preliminary experiments show that it improves efficiency as well as the compactness and quality of the explicit representation of the solution sets.

## 2 Background and Motivation

We address the issue of solving non-linear NCSPs with *continuum of solutions* (Figure 1). In its most general form, a continuum of solutions expresses a spectrum of equally relevant choices, as the possible moving areas of a mobile robot, the collision regions between objects in mechanical assembly, or different alternatives of shapes for the components of a kinematic chain. These alternatives need to be identified as precisely and completely as possible. Interval constraint-based solvers (e.g. Numerica [5], ILOG Solver [6]) take as input a numerical CSP, where the domains of the variables are intervals over the reals, and generate a set of boxes which *conservatively* enclose each solution (no solution is lost). They have proven particularly efficient in solving challenging instances of numerical CSPs with non-linear constraints but are commonly designed to deliver point-wise solutions. As a consequence, when applied to our target problems, the approximations they provide for the complete solution set are, in most cases, prohibitively verbose. As an example, let us consider the following NCSP with four non-linear inequality constraints and three variables: $P3 = \{x^2 \leq y, \ln y + 1 \geq z, xz \leq 1, x^{3/2} + \ln(1.5z + 1) \leq y + 1, x \in [-15,\ 15], y \in [1,\ 200], z \in [-10,\ 10]\}$.

Using an efficient implementation of classical point-wise techniques,[1] the computation had to be stopped after 1 hour and produced more than 90000 small boxes.[2] A natural alternative to the point-wise approach is to try to cover the spectrum of non-isolated solutions using a *reduced* number of subsets from $\mathbb{R}^n$. Usually, these subsets are chosen with known and simple properties (e.g. interval boxes, polytopes, ellipsoids) [7]. In recent years, several authors have proposed set covering algorithms with interval boxes [7, 8,9,10]. These algorithms are based on domain splitting and have one of the following limitations: they are designed for inequality constraints only [8,9,10], they only apply to polynomials [9], they uniformly enforce dichotomous splitting on all variables [7]. Moreover, most of these techniques produce verbose approximations of the boundaries [7,8, 9]. As a consequence, either their applicability is restricted or the tractability limits are rapidly reached. The alternative technique we propose is based on the following observations. Firstly, when there are non-isolated solutions, dichotomous splitting is not the most adapted branching strategy. It might lead to unnecessarily dividing entirely feasible regions. We propose an alternative scheme based on splitting around the negation of feasible regions. Secondly, the union of boxes produced by the complete solving of numerical CSPs with continuum of solutions can be seen as an orthogonal polyhedron. Enhanced representations from computational geometry can be used to reduce the verbosity of such geometrical object. We propose to use the *extreme vertex representation* of orthogonal polyhedra [1,2,3] for this purpose.

## 3 Definitions and Notations

### 3.1 Interval Arithmetic

The finite nature of computers precludes an exact representation of the reals. The set $\mathbb{R}$, extended with the two infinity symbols, and then denoted by $\mathbb{R}_\infty = \mathbb{R} \cup \{-\infty, +\infty\}$, is in practice approximated by a finite subset $\mathbb{F}_\infty$ containing $-\infty$, $+\infty$ and 0. In interval-based constraint solvers, $\mathbb{F}_\infty$ usually corresponds to the floating-point numbers used in the implementation. Let $<$ be the natural extension to $\mathbb{R}_\infty$ of the order relation $<$ over $\mathbb{R}$. For each $l$ in $\mathbb{F}_\infty$, we denote by $l^+$ the smallest element in $\mathbb{F}_\infty$ greater than $l$, and by $l^-$ the greatest element in $\mathbb{F}_\infty$ smaller than $l$.

The set of intervals with bounds in $\mathbb{F}_\infty$, denoted by $\mathbb{I}$, is ordered by set inclusion. In the rest of the paper, intervals are written uppercase, reals or floats are lowercase, vectors in boldface and sets in uppercase calligraphic letters. An *interval box* (henceforth referred to *box*) $\mathbf{B} = I_1 \times \ldots \times I_n$ is a Cartesian product of $n$ intervals in $\mathbb{I}$. A *canonical interval* is a non-empty interval of the form $[l..l]$ or of the form $[l..l^+]$. A *canonical box* is a Cartesian product of canonical intervals.

We use the following notations for set theoretic operations on boxes: $\mathbf{A} \sqcup \mathbf{B} = \mathbf{A} \cup \mathbf{B}$, $\mathbf{A} \sqcap \mathbf{B} = cl(int(\mathbf{A}) \cap int(\mathbf{B}))$, $\neg\mathbf{A} = cl(\sim \mathbf{A})$, where $cl$ and $int$ are the topological closure and interior operations, and $\sim$ is the complementary operation.

[1] The implementation was based on ILOG solver 5.1 (see Section 6).

[2] The alternative technique we propose could reduce the complete output to 1373 boxes and produced the result in 5.63 seconds (see Table 1).

### 3.2 Relations and Approximations

Let $c(x_1, \ldots, x_n)$ be a real constraint with arity $n$. The *relation* defined by $c$, denoted by $\rho_c$, is the set of tuples satisfying $c$. The relation defined by the negation, $\neg c$, of $c$ is given by $\mathbb{R}^n \setminus \rho_c$. The global *relation* defined by the conjunction of all the constraints of a NCSP, $\mathcal{C}$ is denoted $\rho_{\mathcal{C}}$. It can be approximated by a computer-representable superset or subset. In the first case the approximation is *complete* but may contain points that are not solutions. Conversely, in the second case, the approximation is *sound* but may lose certain solutions. A relation $\rho$ can be approximated conservatively by the smallest (w.r.t set inclusion) union of boxes, or more coarsely by the smallest box, containing it. In the rest of the paper, we will use the following definitions and notations:

**Definition 1 (The Minimal Outer Box, $\mathbf{OB}$).** *Let $\rho$ be a relation from $\mathbb{R}^n$, the minimal outer box of $\rho$, denoted by $\mathbf{OB}(\rho)$, is defined by:*

$$\mathbf{OB}(\rho) \in \mathbb{I}^n, \mathbf{OB}(\rho) \supseteq \rho : \forall \mathbf{B} \in \mathbb{I}^n, \mathbf{B} \supseteq \rho \Rightarrow \mathbf{OB}(\rho) \subseteq \mathbf{B} \tag{1}$$

**Definition 2 (The Best Outer Approximation, $\mathbf{OA}$).** *Let $\rho$ be a relation from $\mathbb{R}^n$, the best outer approximation of $\rho$, denoted by $\mathbf{OA}(\rho)$, is defined by:*

$$\mathbf{OA}(\rho) = \bigcup_{r \in \rho} \mathbf{OB}(\{r\}) \tag{2}$$

**Definition 3 (The Best Inner Approximation, $\mathbf{IA}$).** *Let $\rho$ be a relation from $\mathbb{R}^n$, the best inner approximation of $\rho$, denoted by $\mathbf{IA}(\rho)$, is defined by:*

$$\mathbf{IA}(\rho) = \bigcup_{\mathbf{B} \in \mathbb{I}^n, \mathbf{B} \subseteq \rho} \mathbf{B} \tag{3}$$

**Definition 4 (The Best Undiscernible Approximation, $\mathbf{UA}$).** *Let $\rho$ be a relation from $\mathbb{R}^n$, the best undiscernible approximation of $\rho$, denoted by $\mathbf{UA}(\rho)$, is the difference between $\mathbf{OA}(\rho)$ and $\mathbf{IA}(\rho)$: $\mathbf{UA}(\rho) = \mathbf{OA}(\rho) \setminus \mathbf{IA}(\rho)$.*

Figure 2 illustrates Definitions 1, 2, 3 and 4 on a simple example. The relation to approximate contains all the points lying inside the circle and on its boundary.

**Proposition 1.** *Given a relation, $\rho \subseteq \mathbb{R}^n$, these properties hold: (1) $\mathbf{OB}(\rho)$ exists uniquely; (2) $\mathbf{IA}(\rho) \subseteq \rho \subseteq \mathbf{OA}(\rho)$; (3) $\mathbf{OA}(\rho)$, $\mathbf{UA}(\rho)$ are minimal and $\mathbf{IA}(\rho)$ is maximal, i.e. $\forall \mathcal{S}_i, \mathcal{S}_o \in \mathcal{P}(\mathbb{I}^n)$, $\mathcal{U}_i = \bigcup_{\mathbf{B} \in \mathcal{S}_i} \mathbf{B}$, $\mathcal{U}_o = \bigcup_{\mathbf{B} \in \mathcal{S}_o} \mathbf{B}$:*

$$\mathcal{U}_i \subseteq \rho \subseteq \mathcal{U}_o \Rightarrow \mathcal{U}_i \subseteq \mathbf{IA}(\rho) \subseteq \mathbf{OA}(\rho) \subseteq \mathcal{U}_o, \mathbf{UA}(\rho) \subseteq \mathcal{U}_o \setminus \mathcal{U}_i \tag{4}$$

The computation of these approximations relies on the notion of *contracting operators*. Basically, a contracting operator narrows down the variable domains by discarding values that are locally inconsistent. In this paper we use the notion of outer-bound contracting operator, defined as follows:

**Definition 5 (Outer-bound Contracting Operator, $\mathbf{OC}$).** *An outer-bound contracting operator is a function $\mathbf{OC} : \mathbb{I}^n \times \mathcal{P}(\mathbb{R}^n) \rightarrow \mathbb{I}^n$ such that $\forall \mathbf{B} \in \mathbb{I}^n$, $\rho \in \mathcal{P}(\mathbb{R}^n)$ these properties hold:* [3]

[3] $\mathcal{P}(S)$ denotes the power-set of $S$, i.e., the set $\{A | A \subseteq S\}$.

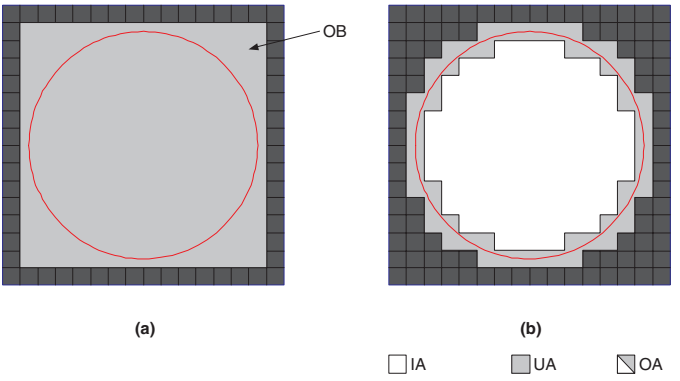



**Fig. 2.** The Best Approximations: (a) $\mathbf{OB}$; (b) $\mathbf{IA}$ (white), $\mathbf{UA}$ (gray) and $\mathbf{OA}$ (white & gray). The relation to approximate contains all the points lying inside the circle and on its boundary

$$(1)\ \mathbf{OC}(\mathbf{B},\rho) \subseteq \mathbf{B} \quad \textit{(Contractiveness)}$$
$$(2)\ \mathbf{OC}(\mathbf{B},\rho) \supseteq \mathbf{B} \cap \rho \ \textit{(Completeness)}$$

Often, a monotonicity condition is also required to guarantee confluence. We do not consider this restriction for the moment. In numerical domains, the outer-bound contracting operators usually enforce either *Box*, *Hull*, *kB* or *Bound* consistency [11,5], generally referred to as bound-consistency in the rest of the paper.

### 3.3 Union Approximations

In this paper we consider the problem of computing $\mathbf{IA}(\rho)$ and $\mathbf{OA}(\rho)$ approximations of a relation $\rho \subseteq \mathbb{R}^n$ under the form of *unions of disjoint boxes*[4].

**Definition 6 (Outer Union Approximation, $\mathbf{Union}^{\mathcal{O}}(\rho)$).** $\mathbf{Union}^{\mathcal{O}}(\rho)$ *is a set of disjoint boxes* $\mathcal{U} \in \mathcal{P}(\mathbb{I}^n)$ *such that:*

$$\bigcup_{\mathbf{B}\in\mathcal{U}} \mathbf{B} \supseteq \mathbf{OA}(\rho) \tag{5}$$

**Definition 7 (Inner Union Approximation, $\mathbf{Union}^{\mathcal{I}}(\rho)$).** $\mathbf{Union}^{\mathcal{I}}(\rho)$ *is a set of disjoint boxes* $\mathcal{U} \in \mathcal{P}(\mathbb{I}^n)$ *such that:*

$$\bigcup_{\mathbf{B}\in\mathcal{U}} \mathbf{B} \subseteq \mathbf{IA}(\rho) \tag{6}$$

**Definition 8 (Undiscernible Union Approximation, $\mathbf{Union}^{\mathcal{U}}(\rho)$).** $\mathbf{Union}^{\mathcal{U}}(\rho)$ *is a set of disjoint boxes* $\mathcal{U} \in \mathcal{P}(\mathbb{I}^n)$ *such that:*[5]

$$\bigcup_{\mathbf{B}\in\mathcal{U}} \mathbf{B} = cl\Big(\bigcup_{\mathbf{B}\in\mathbf{Union}^{\mathcal{O}}(\rho)} \mathbf{B} \setminus \bigcup_{\mathbf{B}\in\mathbf{Union}^{\mathcal{I}}(\rho)} \mathbf{B}\Big) \tag{7}$$

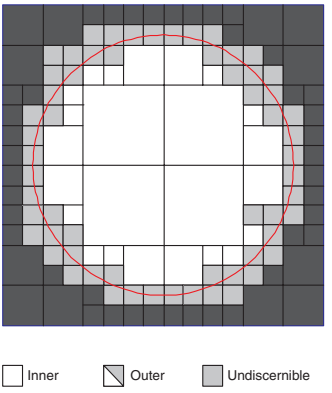



**Fig. 3.** The Union Approximations: $\mathbf{Union}^{\mathcal{O}}(\rho)$ (white & gray), $\mathbf{Union}^{\mathcal{I}}(\rho)$ (white) and $\mathbf{Union}^{\mathcal{U}}(\rho)$ (gray)

Figure 3 illustrates these definitions on the example of Figure 2.

Several authors have recently addressed the issue of computing $\mathbf{Union}^{\mathcal{O}}$ approximations. In [7], a recursive dichotomous split is performed on the variable domains. Each box obtained by splitting is tested for inclusion using interval arithmetic tools. The boxes obtained are hierarchically structured as $2^k$-*trees*. The authors have demonstrated the practical usefulness of such techniques in robotics, etc. In [8], a similar algorithm is presented. However, only binary or ternary subsets of variables are considered when performing the splits. The approach is restricted to classes of problems with convexity properties. The technique proposed in [9] constructs the union algebraically using Bernstein polynomials, which makes it possible to use guaranteed inclusion tests for boxes. The approach is restricted to polynomial constraints. A technique to extend consistent domains of particular class of constraints has also been proposed in [12]. Finally, [10] has addressed the issue of computing $\mathbf{Union}^{\mathcal{I}}$ approximations for universally quantified constraints.

## 4 Back-Boxing and EVR

Interval-based search techniques for NCSPs are essentially dichotomous. Variables are instantiated using intervals. When the search reaches an interval that contains no solutions it backtracks, otherwise the interval is recursively split into two halves up to an established resolution. The most successful techniques enhance this process by applying an outer-bound contracting operator to the overall constraint system, after each split. In all the known algorithms, the general policy is to perform splitting until canonical intervals are reached and as long as the error inherent to the outer-bound contracting operator is smaller than the interval to split. This policy, referred to as `DMBC` (dichotomous maintaining bound-consistency) in the rest of the paper, works generally well for systems

[4] Two boxes, $\mathbf{B_1}$ and $\mathbf{B_2}$, are said disjoint if $\mathbf{B_1} \not= \mathbf{B_2} \Rightarrow \mathbf{B_1} \sqcap \mathbf{B_2} = \emptyset$.

[5] Informally, $\mathbf{Union}^{\mathcal{U}}(\rho)$ is a set of undiscernible boxes enclosing the boundary of $\rho$.

with isolated solutions but leaves room for improvement when there is a continuum of feasible points. The improvements we propose are presented in the two next subsections.

### 4.1 Better Splitting Decisions Using Back-Boxing

In order to reduce as much as possible the number of disjoint boxes required for covering the solution space, we first try to avoid the split of completely feasible boxes. To achieve this goal, we use a feasibility (soundness) test for boxes [10],[6] which allows better splitting decisions. Given a relation, $\rho$, and a box, $\mathbf{B}$, the feasibility test checks whether the whole box is contained in $\rho$ or not. It is based on the following obvious property: $\mathbf{B} \cap \neg\rho = \emptyset \Leftrightarrow \mathbf{B} \subseteq \rho$. We use this property to implement a contracting operator, called *back-boxing contracting* operator and a splitting operator, called *back-box splitting* operator.

**Definition 9 (Back-Boxing Contracting Operator, $\mathbf{BBC}$).** *A back-boxing contracting operator w.r.t. an* $\mathbf{OC}$ *operator is a function* $\mathbf{BBC} : \mathbb{I}^n \times \mathcal{P}(\mathbb{R}^n) \rightarrow \mathbb{I}^n$ *such that*

$$\forall \mathbf{B} \in \mathbb{I}^n, \rho \in \mathcal{P}(\mathbb{R}^n) : \mathbf{BBC}(\mathbf{B}, \rho) = \mathbf{OC}(\mathbf{B}, \neg\rho)$$

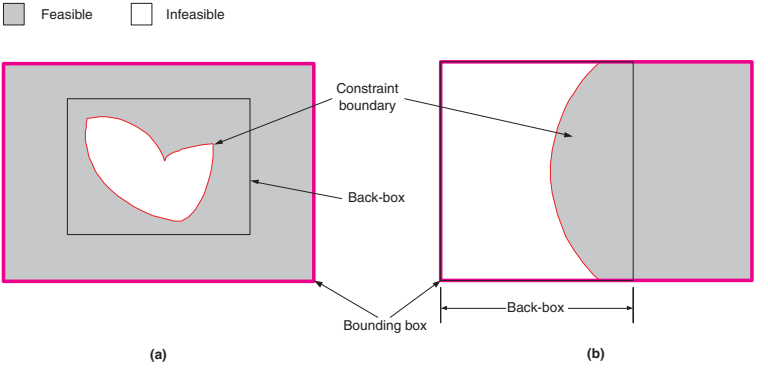



**Fig. 4.** Two examples of back-boxing contractions

For simplicity, given a finite set of constraints $\mathcal{C} = \{c_1, ..., c_n\}$, we denote $\mathbf{BBC}(\mathbf{B}, \rho_C)$ by $\mathbf{BBC}(\mathbf{B}, c_1, ..., c_n)$ or $\mathbf{BBC}(\mathbf{B}, \mathcal{C})$.

The following properties characterize back-box contracting operators.

**Proposition 2.** *Given a relation,* $\rho$*, and a box,* $\mathbf{B}$*, if there is some* $\mathbf{BBC}$ *operator that contracts* $(\mathbf{B}, \rho)$ *to an empty set, then* $\mathbf{B}$ *is completely contained in* $\rho$*, i.e.*

$$\exists \mathbf{BBC} : \mathbf{BBC}(\mathbf{B}, \rho) = \emptyset \Rightarrow \mathbf{B} \subseteq \rho$$

[6] In [10] such a feasibility test is used for individual constraints. We use it for conjunction of constraints.

*Proof.* We have $\mathbf{BBC}(\mathbf{B},\rho) = \emptyset$, $\mathbf{BBC}(\mathbf{B},\rho) = \mathbf{OC}(\mathbf{B},\neg\rho)$, and $\mathbf{OC}(\mathbf{B},\neg\rho) \supseteq \mathbf{B} \cap \neg\rho$, then $\mathbf{B} \cap \neg\rho = \emptyset$ , this implies that $\mathbf{B} \subseteq \rho$.

**Corollary 1.** *Given a finite set of constraints,* $\mathcal{C} = \{c_1, ..., c_n\}$*, and a bounding box,* $\mathbf{B}$*. The box* $\mathbf{B}$ *is completely feasible (w.r.t* $\mathcal{C}$*) if there is some* $\mathbf{BBC}$ *operator that contracts* $(\mathbf{B},\mathcal{C})$ *to an empty set, i.e.*

$$\exists \mathbf{BBC} : \mathbf{BBC}(\mathbf{B},\mathcal{C}) = \emptyset \Rightarrow \mathbf{B} \textit{ is feasible (w.r.t } \mathcal{C})$$

This corollary implies that back-boxing makes it possible to isolate completely feasible boxes with respect to some constraints. Figure 4 illustrates the behavior of a back-boxing contracting operator. A back-box results from the application of a $\mathbf{BBC}$ operator to a box $\mathbf{B}$ and a relation $\rho_c$ identified by a constraint $c$. When applying a back-boxing contracting operator to a box with respect to a constraint results in a empty set, it can be deduced that the box completely satisfies that constraint. Similarly, when applying a back-boxing contracting operator to a box with respect to the whole constraint set results in a empty set, the box can be stated as completely feasible. We then define a splitting operator based on back-boxing, which consists of splitting around back-boxes:

**Definition 10 (Back-Box Splitting Operator: BBS).** *A back-box splitting operator is a function* $\mathbf{BBS} : \mathbb{I}^n \times \mathbb{I}^n \rightarrow \mathcal{P}(\mathbb{I}^n)$ *splitting a bounding box,* $\mathbf{B}$*, along the faces of a back-box,* $\mathbf{BB}$*.*

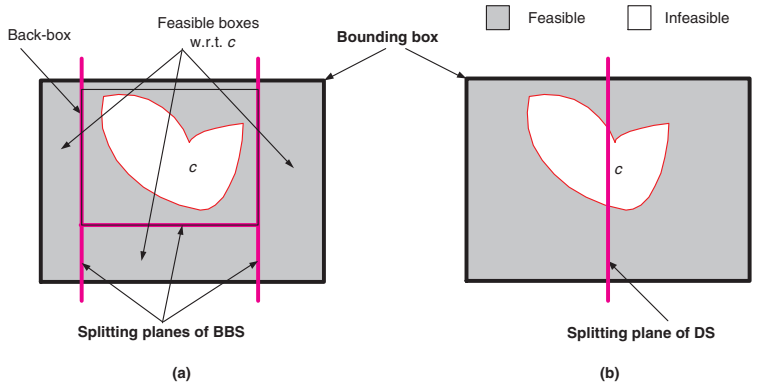



**Fig. 5.** (a) Back-Box Splitting: splitting around back-boxes; (b) Dichotomous Splitting: splitting the original domain of a variable into two halves

In the algorithms we propose, back-box splitting is applied in combination with dichotomous splitting. The latter is used either when back-boxing produces no reduction or when back-box splitting results in too small boxes (i.e. it is performed close to the boundaries). Figure 5 illustrates the notion of back-box splitting.

### 4.2 Concise Representations of the Boundaries Using EVR

Back-boxing is mainly intended to reduce the number of boxes of *inner* union approximations. We now address the issue of producing more compact *undiscernible* union

approximations. Box-covering, as usually implemented, often produces a significant number of nearly aligned boxes along the boundary of the constraints. These boxes introduce artificial convexity deficiencies which are only due to the orthogonal splitting policy.

**Better Alignment of Boxes.** We observe that a better alignment can be obtained by closely controlling the application of the contracting operators during search. More precisely, whenever some dimension, $i$, of a box, $\mathbf{B}$, reaches precision $\varepsilon_i$, one can prevent the contracting operator to contract $\mathbf{B}$ over this dimension in order to obtain better alignments and performances.

**Definition 11 (Active/Inactive Dimension).** *Given a box, $\mathbf{B}$, a set of constraints, $\mathcal{C}$, and a precision vector, $\varepsilon$. A dimension, $i$, of $\mathbf{B}$ is called* active dimension *if the size of $\mathbf{B}$ in dimension $i$ exceeds $\varepsilon_i$ and if the corresponding variable, $v_i$, occurs in some constraint of $\mathcal{C}$. Otherwise, it is said to be an* inactive dimension*.*

A contracting operator working on the active dimensions of a box only will be called a *restricted-dimensional contracting operator*. Hereafter, we denote as $\mathbf{OC_{rd}}$ (respectively, $\mathbf{BBC_{rd}}$) the restricted-dimensional contracting operators corresponding to $\mathbf{OC}$ (respectively, $\mathbf{BBC}$) operators. There are several ways of implementing $\mathbf{OC_{rd}}$ and $\mathbf{BBC_{rd}}$ depending on which $\mathbf{OC}$ operator is used. The first way, consists of using some classical $\mathbf{OC}$ operators working in full-dimension. After a box $\mathbf{B}$ has been contracted over all dimensions, the inactive dimensions are simply restored to their original sizes. The gain is then only a better alignment of boxes. The second way consists of using an $\mathbf{OC}$ operators which directly allows a restricted-dimensional contracting.[7] A box $\mathbf{B}$ will then only be contracted over the active dimensions. Such an $\mathbf{OC}$ operator does not require the returned box to be bound-consistent in the inactive dimensions. The returned box will therefore be usually larger than the one returned by an $\mathbf{OC}$ operator working in full dimension. However, not only better alignments can be obtained but also better performances. The third possible way consists of applying the $\mathbf{OC}$ operator to the *projection* of the relation (or constraints) over the active dimensions only. The result is composed back to the full-dimensional box by adding the inactive dimensions with the data of the original box. This alternative can only be used for the constraints that can be easily projected symbolically.

**Compacting Aligned Boxes.** Once a better alignment is obtained, the question is how such a set of aligned boxes can be compacted into a smaller set of boxes. We propose to use the Extreme Vertex Representation of orthogonal polyhedra for that purpose. The basic idea is that the finite unions of boxes delivered by box-covering solver define *orthogonal polyhedra* for which improved representations can be used. Informally, *orthogonal polyhedra* are the ones whose facets are axis-parallel hyper-rectangles. They can be naturally represented as a finite union of disjoint boxes. Such a representation is called the *Disjoint Box Representation* (DBR) in computational geometry. The *Extreme Vertex Representation* (EVR) is a way of compacting a DBR representation. It was first proposed in [1] for 3-dimensional orthogonal polyhedra and generalized to n-dimensional orthogonal polyhedra in [2,3]. We now recall some basic concepts related to EVR. We refer the reader to [2,3] for further details. The concepts we present

[7] ILOG Solver 5.1 provides such $\mathbf{OC_{rd}}$ operators. We use them in our implementation.

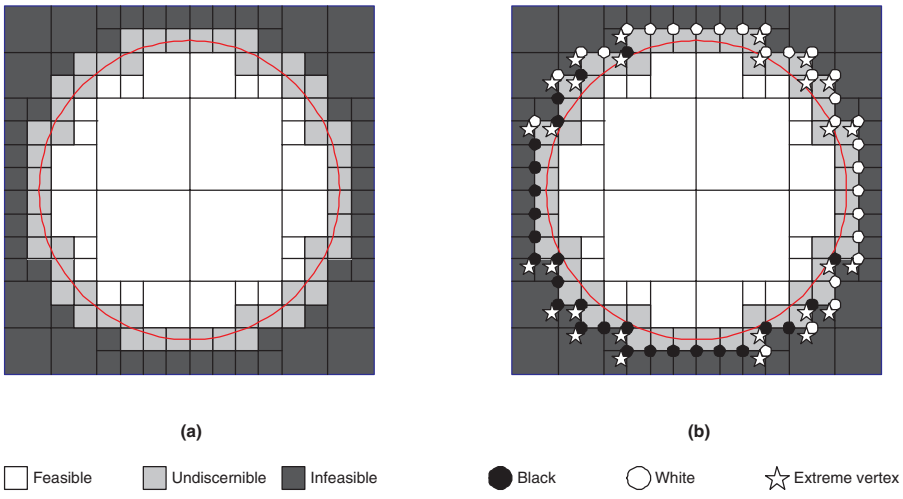



**Fig. 6.** (a) DBR of union approximations; (b) EVR of $\mathbf{Union}^{\mathcal{O}}$

relate to a particular type of orthogonal polyhedra, called *griddy polyhedra*. Informally, a griddy polyhedron [3] is generated from unit hyper-cubes with integer-valued vertices. Since arbitrary orthogonal polyhedra can be obtained from griddy ones by appropriate stretching and translation, the results on EVR are not affected by this simplification. In fact they even do not depend on an orthogonal basis. For simplicity, we assume that the polyhedra live inside a bounded subset $\mathbf{X} = [0, m]^d \subseteq \mathbb{R}^d$ (in fact, the results will hold also for $\mathbf{X} = \mathbb{R}^d_+$). Let $\mathbf{x} = (x_1, ..., x_d)$ be a grid point of the elementary grid $\mathcal{G} = \{0, 1, ..., m-1\}^d \subseteq \mathbb{N}^d$. For every point $\mathbf{x} \in \mathbf{X}$, $\lfloor \mathbf{x} \rfloor$ is the grid point corresponding to the integer part of the components of $\mathbf{x}$. The elementary box associated with $\mathbf{x}$ is the closed subset of $\mathbf{X}$ of the form $\mathbf{B}(\mathbf{x}) = [x_1, x_1 + 1] \times ... \times [x_d, x_d + 1]$. The set of all boxes is denoted by $\mathcal{B}$. A griddy polyhedron $P$ is a union of elementary boxes, i.e. an elementary of $2^{\mathcal{B}}$.

**Definition 12 (Color Function).** *Let $P$ be a griddy polyhedron. The color function $c$: $\mathbf{X} \to \{0, 1\}$ is defined as follows: if $\mathbf{x}$ is a grid point then $c(\mathbf{x}) = 1$ iff $\mathbf{B}(\mathbf{x}) \subseteq P$; otherwise, $c(\mathbf{x}) = c(\lfloor \mathbf{x} \rfloor)$.*

We say that a grid point $\mathbf{x}$ is black (respectively, white) and that $\mathbf{B}(\mathbf{x})$ is full (respectively, empty) when $c(\mathbf{x}) = 1$ (respectively 0). A *canonical representation scheme* for $2^{\mathcal{B}}$ (or $2^{\mathcal{G}}$) is a set $\mathcal{E}$ of syntactic objects such that there is some bijective function $\Psi : \mathcal{E} \to 2^{\mathcal{B}}$.

**Definition 13 (Extreme Vertex).** *A grid point $\mathbf{x}$ is said to be extreme if $\tau(\mathbf{x}) = 1$, where $\tau(\mathbf{x})$ denotes the parity of the number of black grid points in $\mathcal{N}(\mathbf{x}) = \{x_1 - 1, x_1\} \times ... \times \{x_d - 1, x_d\}$ (the neighborhood of $\mathbf{x}$).*

Figure 6 illustrates the notion of EVR on a simple example. The fundamental theorem presented in [2,3] shows that any griddy polyhedron can be canonically represented by the set of its extreme vertices and their colors. The *extreme vertex representation*

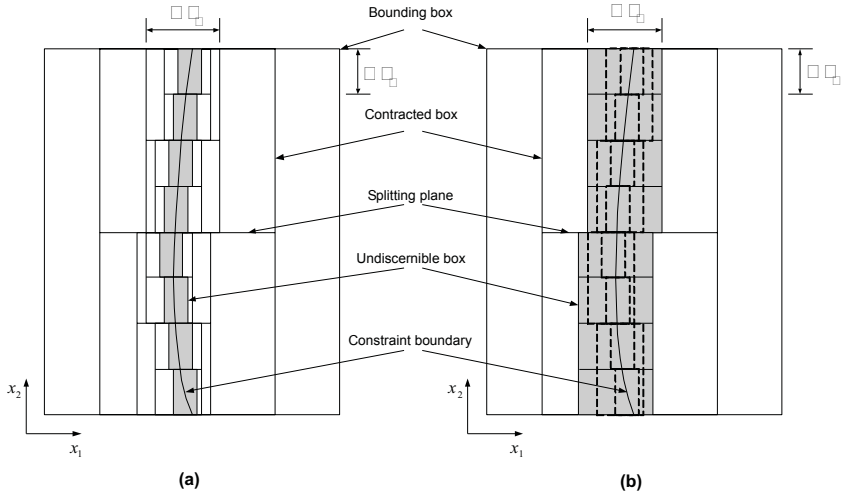



**Fig. 7.** Constraint boundary in a bounding box: (a) unaligned boxes produced by standard covering; (b) enlarged and aligned boxes using EVR

improves the space required for storing orthogonal polyhedra by an order of magnitude [3]. It also enables the design of efficient algorithms for fundamental operations on orthogonal polyhedra (e.g. membership, set-theoretic operations). In particular, effective transformation between DBR and EVR can be proposed for low dimension and/or small size (i.e. $m$ is small) polyhedron [1]. For example, in three dimensions, the average experimental (time) complexity of converting an EVR to a DBR is far less than quadratic but slightly greater than linear in the number of extreme vertices [1]. Results in [3] also imply that, for a fixed dimension, the time complexity of converting a DBR to an EVR using XOR operations is linear in the number of boxes in DBR. We propose to exploit these effective transformation schemes to produce a compact representation of aligned contiguous boxes using the following procedure:

1. Produce a better alignment of the boxes along the boundary of the constraints. This is done by preventing the unnecessary application of contracting operators over the inactive dimensions. Figure 7 shows the better alignment produced for a set of nearly aligned boxes of an undiscernible approximation. The original set of 8 small boxes (Figure 7(a)) reduces to two groups of 4 aligned boxes (Figure 7(b)) without altering the predefined precision.
2. The set of aligned boxes in each group, $\mathcal{S}_1$, is converted to EVR and then back to DBR to get a set of combined boxes, $\mathcal{S}_2$ (containing only one box in this case). Due to the properties of EVR, this procedure guarantees that $\mathcal{S}_2$ has a more concise size than $\mathcal{S}_1$. Figure 7(b) shows how this conversion procedure reduces the two groups of 4 boxes to two (gray) boxes.

Such a procedure can theoretically be applied in any dimensions. Due to the efficiency of EVR in low dimensions, we however restrict its application to low dimensional or small size sub-regions of the search space in our implementation (see Section 5.2).

## 5 Algorithms

We now present two algorithms that compute outer and inner union approximations for non-linear NCSPs, These algorithms, called `UCA6` and `UCA6-Plus` are referred to as `UCA6*` when they have the same properties/operations. A preliminary version of `UCA6` was presented in [13] we will therefore mainly focus on the `UCA6-Plus` algorithm. Given a NCSP $P = (\mathcal{V}, \mathcal{C}, \mathcal{D})$, $\mathbf{B}$ will denote the bounding box of the relation defined by $P$. Originally, this bounding box is set to $\mathcal{D}$. For convenience, we denote $\mathbf{Union}^{\mathcal{X}}(\mathbf{B} \cap$

**function** $UCA6Plus(\ \mathbf{B}_0, \mathcal{C}_0, \varepsilon, \mathbf{OC_{rd}}, \mathbf{BBC_{rd}}, \mathbf{OC}, \mathbf{BBC}, D_{stop})$
  $\mathcal{S}_{inn} := \emptyset$; $\mathcal{S}_{und} := \emptyset$; WList := $\emptyset$;
  **if** $solveQuickly(\mathbf{B}_0, \mathcal{C}_0, \varepsilon, \mathbf{OC_{rd}}, \mathbf{BBC_{rd}}, \mathbf{OC}, \mathbf{BBC}$, WList, $D_{stop})$ **then return**;
  **while** WList $\neq \emptyset$ **do**
    $< \mathbf{B}, \mathcal{C} > := get$(WList);
    **foreach** $c \in \mathcal{C}$ **do**
      $\mathbf{BB}_c := \mathbf{BBC_{rd}}(\mathbf{B}, c)$;
      **if** $\mathbf{BB}_c = \emptyset$ **then**
        $\mathcal{C} := \mathcal{C} \setminus \{c\}$;
      **end**
    **end**
    **if** $\mathcal{C} = \emptyset$ **then**
      $store(\mathcal{S}_{inn}, \mathbf{B})$;
      **continue**; /* while loop */
    **end**
    Split = $getSplit()$;
    **if** Split = 'BBS' **then**
      $\mathbf{BB}_c := chooseTheBest(\mathbf{B}, \{\mathbf{BB}_c | c \in \mathcal{C}\})$;
      $\mathbf{BB} := enlarge(\mathbf{B}, \mathbf{BB}_c, ZeroPlus)$;
      $< \mathbf{B}_1, \ldots, \mathbf{B}_k > := \mathbf{BBS}(\mathbf{B}, \mathbf{BB})$;
    **else**
      $< \mathbf{B}_1, \ldots, \mathbf{B}_k > := \mathbf{DS}(\mathbf{B})$;
    **end**
    **for** $i = 1$ **to** $k$ **do**
      **if** Split = 'BBS' **and** $\mathbf{B}_i \cap \mathbf{BB}_c = \emptyset$ **then**
        **if** $\mathcal{C} = \{c\}$ **then**
          $store(\mathcal{S}_{inn}, \mathbf{B}_i)$;
          **continue**; /* for loop */
        **else**
          $\mathcal{C} := \mathcal{C} \setminus \{c\}$;
        **end**
      **end**
      $solveQuickly(\mathbf{B}_i, \mathcal{C}, \varepsilon, \mathbf{OC_{rd}}, \mathbf{BBC_{rd}}, \mathbf{OC}, \mathbf{BBC}$, WList, $D_{stop})$;
    **end**
  **end**
**end** /* UCA6Plus */

**Fig. 8.** The `UCA6-plus` algorithm

$\rho_{\mathcal{C}}$) as $\mathbf{Union}^{\mathcal{X}}(\mathbf{B}, \mathcal{C})$, where $\mathcal{X} \in \{\mathcal{O}, \mathcal{I}, \mathcal{U}\}$. UCA6* constructs the approximations $\mathbf{Union}^{\mathcal{I}}(\mathbf{B}, \mathcal{C})$ and $\mathbf{Union}^{\mathcal{U}}(\mathbf{B}, \mathcal{C})$, hence $\mathbf{Union}^{\mathcal{O}}(\mathbf{B}, \mathcal{C})$ can be computed as the union of these two approximations. UCA6* proceeds by repeating three main operations: *(i)* using outer-bound contracting operators to contract the current bounding box to a tighter bounding box;[8] *(ii)* using back-boxing contracting operators to get a list of back-boxes w.r.t. each active constraint and w.r.t the contracted box in *(i)*, the constraints that makes the corresponding back-box empty are removed; finally, *(iii)* combining dichotomous splitting with back-box splitting. When the chosen strategy, at a given moment, is back-box splitting, the **BBS** operator is used to split around the best back-box (details are given later). The constraint corresponding to the chosen back-box is then removed from all the surrounding boxes resulting from the **BBS**.[9] In Figure 8 and 9, $\mathcal{S}_{inn}$ and $\mathcal{S}_{und}$, which are global variables, denote the sets of boxes of $\mathbf{Union}^{\mathcal{I}}(\mathbf{B}_0, \mathcal{C}_0)$ and $\mathbf{Union}^{\mathcal{U}}(\mathbf{B}_0, \mathcal{C}_0)$, respectively. We use a list, WList, to store the sub-problems waiting to be processed. WList can be handled as a queue or a stack. This allows for breadth-first search in the former case and to depth-first search in the latter. $chooseTheBest(\mathbf{B}, \{\mathbf{BB}_c | c \in \mathcal{C}\})$ is a function choosing the best back-box and the respective constraint based on some criteria to maximize the space surrounding the back-box. $enlarge(\mathbf{B}, \mathbf{BB}_c, ZeroPlus)$ is a function extending $\mathbf{BB}_c$ to $\mathbf{BB}$ by $ZeroPlus$ (considered as a sufficiently small positive number) such that the result is still in $\mathbf{B}$. This will guarantee that no point satisfying $c$ is on the boundary of $\mathbf{BB}$ except the points on the boundary of $\mathbf{B}$. Figures 8 and 9 give the algorithm UCA6-plus.

### 5.1 Splitting Strategies

$getSplit()$ is a function which returns the splitting mode to be used for splitting the current box. UCA6* uses a combination of the **BBS** and **DS** operators. The current splitting mode returned by $getSplit()$ is inferred from information on the history of the current box. The simplest implementation uses the information concerning the splitting mode of the parent box, for example whether or not the current box is a back-box obtained from splitting the parent box.

In contrast to DMBC, the **DS** operator used for UCA6* only tries to dichotomize over the active dimensions. This avoids splitting boxes into a huge number of tiny boxes. Moreover, in UCA6* constraints are removed gradually whenever an empty back-box is computed w.r.t. those constraints. The dimension with the greatest size is preferred for **DS** split. For the pruning to be efficient, **BBS** splits along some face of a back-box only if the splitting plane produces sufficiently large boxes, the back-box itself excepted. This estimation is done using a pre-determined *fragmentation ratio*.

### 5.2 Applying EVR

The function $solveQuickly$ (Figure 9) is of the main novelties in UCA6-Plus w.r.t. UCA6. This function constructs $\mathbf{Union}^{\mathcal{I}}$ and $\mathbf{Union}^{\mathcal{U}}$ approximations for low-dimensional sub-problems with at most $D_{stop}$ active dimensions. The output is compacted using

[8] Standard operator for UCA6 and restricted-dimensional one for UCA6-Plus.

[9] These boxes are known to be feasible due to the properties of back-boxing.

EVR. The second novelty lies in the fact that UCA6-Plus uses $\mathbf{OC_{rd}}$ and $\mathbf{BBC_{rd}}$ instead of $\mathbf{OC}$ and $\mathbf{BBC}$ for the purpose of narrowing.[10] By doing so UCA6-Plus gains in performance and produces a better alignment of boxes along the boundaries. This allows for using EVR to combine the aligned contiguous boxes. $solveQuickly$ (Figure 9) proceeds by using the following tests: *(i)* check if the bounding box is infeasible; *(ii)* check if the contracted box has no active dimension, then check if it is feasible or undiscernible w.r.t. $\mathcal{C}$; *(iii)* check if the contracted box has at most $D_{stop}$ active dimensions, if so, use an appropriate technique to solve the sub-problem in that box using restricted-dimensional contracting operators, *(iv)* Otherwise, the contracted box is put into the waiting list to be further processed. For efficiency purposes, $solveQuickly$ allows resorting to a secondary search technique, $DimStopSolver$, to solve the low-dimensional sub-problems whose bounding box has at most $D_{stop}$ active dimensions. Good candidates for small $D_{stop}$ can be either the $2^k$-tree based solver presented in [8] or a simple grid-based solver[11]. Variants of DMBC or UCA6 using the restricted-dimensional contracting operators can alternatively be used. For a given sub-problem, $DimStopSolver$ constructs the sets $\mathcal{S}'_{inn}$ and $\mathcal{S}'_{und}$ which are the $\mathbf{Union}^{\mathcal{I}}$ and $\mathbf{Union}^{\mathcal{U}}$ of the sub-problem, respectively. These two sets are represented in DBR. They are converted to EVR and then back to DBR to combine each group of aligned contiguous boxes into a bigger equivalent box. This operation is represented by the function $combine$ in Figure 9. The results after combination are stored in the set of boxes of $\mathbf{Union}^{\mathcal{I}}(\mathbf{B}_0, \mathcal{C}_0)$ and $\mathbf{Union}^{\mathcal{U}}(\mathbf{B}_0, \mathcal{C}_0)$.

**Proposition 3.** *Let $P = (\mathcal{V}, \mathcal{C}, \mathcal{D})$ be a numeric CSP, the* UCA6 *and* UCA6-Plus *algorithms compute an inner union approximation ($\mathbf{Union}^{\mathcal{I}}$) and outer union approximation ($\mathbf{Union}^{\mathcal{O}}$) for $\rho_{\mathcal{C}}$ w.r.t. the predefined precision.*

*Sketch of proof*: The rigorous proof of this proposition is out of scope of the paper. However, we can see informally that given the properties of the contracting operators used for these algorithm: *(i)* the outer-bound contracting operator always produces a box which contains the active relation, and *(ii)* the back-boxing contracting operator always produces a back-box which contains the negation of the active relation, then it is safe to remove the active relation (defined by one or many constraints) from all the regions surrounding the back-box and contained in the bounding box.

## 6 Preliminary Experiments

Only a small amount of work exists on computing inner and outer union approximations for numerical CSPs with non-linear constraints. Often these problems are recast as optimization problems, with artificial optimization criteria, to fit the solvers. Hence, no significant set of benchmarks is presently available in this area. In this section we present a preliminary evaluation on the following small set of typical problems (with different types of solution space).

$CD$ (column design) and $FD$ (fatigue design) are two engineering design examples. Their complete descriptions are available at *http://imacsg4.epfl.ch:8080/PGSL/*. In Table 1, the considered instance of *CD* is the one that finds $(a, b, e) \in [0.01, 2] \times [0.01, 1] \times$

[10] $\mathbf{OC}$ and $\mathbf{BBC}$ are still used for checking the feasibility of $\varepsilon$-bounded boxes.

[11] A simple grid-based solver splits the variable domains into a grid and then solves the problem in each grid element.

**function** $solveQuickly(\mathbf{B}, \mathcal{C}, \varepsilon, \mathbf{OC_{rd}}, \mathbf{BBC_{rd}}, \mathbf{OC}, \mathbf{BBC}$, WList, $D_{stop})$
  **if** $\mathbf{B}$ *has no active dimension* **then**
    $\mathbf{B}' := \mathbf{OC}(\mathbf{B}, \mathcal{C})$;
    **if** $\mathbf{B}' = \emptyset$ **then return** True;
    **if** $\mathbf{BBC}(\mathbf{B}, \mathcal{C}) = \emptyset$ **then**
      $store(\mathcal{S}_{inn}, \mathbf{B})$;
      **return** True;
    **end**
    $store(\mathcal{S}_{und}, \mathbf{B})$;
    **return** True;
  **end**
  $\mathbf{B}' := \mathbf{OC_{rd}}(\mathbf{B}, \mathcal{C})$;
  **if** $\mathbf{B}' = \emptyset$ **then return** True;
  **if** $\mathbf{B}'$ *has no active dimension* **then**
    **if** $\mathbf{BBC}(\mathbf{B}', \mathcal{C}) = \emptyset$ **then**
      $store(\mathcal{S}_{inn}, \mathbf{B}')$;
      **return** True;
    **end**
    $store(\mathcal{S}_{und}, \mathbf{B}')$;
    **return** True;
  **end**
  **if** $\mathbf{B}'$ *has at most* $D_{stop}$ *active dimensions* **then**
    $< \mathcal{S}'_{inn}, \mathcal{S}'_{und} > := DimStopSolver(\mathbf{B}', \mathcal{C}, \varepsilon, \mathbf{OC_{rd}}, \mathbf{BBC_{rd}}, \mathbf{OC}, \mathbf{BBC})$;
    $store(\mathcal{S}_{inn}, combine(\mathcal{S}'_{inn}))$;
    $store(\mathcal{S}_{und}, combine(\mathcal{S}'_{und}))$;
    **return** True;
  **end**
  $put$(WList,$< \mathbf{B}', \mathcal{C} >$);
  **return** False;
**end** /* solveQuickly */

**Fig. 9.** The function $SolveQuickly$

$[0.05, 0.1]$ given the eccentric load $P = 400kN$, the height of column $H = 6m$ and the eccentricity load $L = 1m$, where $a$ and $b$ denote the width and depth of the column cross section (in meter) respectively, $e$ the thickness (in decimeter). The $FD$ instance considered is the one that finds $(L, qf, Z) \in [10, 30] \times [70, 90] \times [0.1, 10]$ for a given number of years to fatigue failure $years = 100$, where $qf$ is the permissible load and $Z$ is the section modulus (scaled up 100 times in unit).

$WP$ is a 2D simplification of the design model for a kinematic pair consisting of a wheel and a pawl. The constraints determine the regions where the pawl can touch the wheel without blocking its motion.
$WP = \{20 < \sqrt{x^2 + y^2} < 50,\ 12y/\sqrt{(x-12)^2 + y^2} < 10,\ x \in [-50,\ 50],\ y \in [0,\ 50]\}$.

$SB$ describes structural and safety restrictions for the components of a floor consisting of a concrete slab on steel beams.
$SB = \{u + c_1 w^{1.5161} - p = 0,\ u - (c_6 h_s + c_7)s \leq 0, c_2 - c_3 s + c_4 s^2 - c_5 s^3 - h_s \leq 0,$

$c_8(pw^2)^{0.3976} - h_b \leq 0$, $c_9(pw^3)^{0.2839} - h_b$, $w \in [5800,\ 13500]$, $p \in [15,\ 50]$, $h_b \in [150,\ 223]$, $s \in [1900,\ 2200]$, $u \in [10,\ 60]$, $h_s \in [75,\ 200]\}$.

$P1 = \{x_0 = x_1 + 1, x_2 + 1 = x_0 + x_1, x_2 \geq x_0 + 2, x_1 + 2x_3 \geq x_4, x_2 - x_3 \leq 3, x_i \in [-10,\ 10]\ \}$.

$P2 = \{x^2 \leq y, \ln y + 1 \geq z, xz \leq 1, x \in [-15,\ 15], y \in [1,\ 200], z \in [-10,\ 10]\}$.

$P3 = \{x^2 \leq y, \ln y + 1 \geq z, xz \leq 1, x^{3/2} + \ln(1.5z + 1) \leq y + 1, x \in [-15,\ 15], y \in [1,\ 200], z \in [-10,\ 10]\}$.

For comparison and evaluation purposes, we have implemented the algorithms `DMBC`, `UCA6`, `UCA6-Plus` using the same data structure and the same standard contracting operators. We have also implemented a version of `DMBC` including the feasibility test. This enhanced version, called `DMBC+`, can therefore check whether a box is completely feasible or not. Our experiments discard `DMBC` as a reasonable candidate for this kind of problems. It always produces a huge number of boxes, each of which is $\varepsilon$-bounded. The tests shown in Table 1 were run with a fragmentation ratio of 0.25 and $D_{stop}$ = 1. The standard **OC** operator was implemented with ILOG Solver 5.1 [6]. Each cell in the table has two rows. The first shows time and the second the number of boxes in the $\mathbf{Union}^{\mathcal{I}}$ and $\mathbf{Union}^{\mathcal{U}}$ respectively. $Bprec$ is the precision for the contracting operators, $prec$ is the precision of the algorithms. The secondary search technique used for `UCA6-Plus` in Table 1 is the simple grid-based one.

**Table 1.** Test results

| Problem | $prec$ | $Bprec$ | `DMBC+` | `UCA6` | `UCA6-Plus` |
|---|---|---|---|---|---|
| $P1$ | 0.01 | 0.1 | $>$ 1h<br>$>$ 0/90000 | 94.89s<br>0/11465 | 66.17s<br>0/6415 |
| $P2$ | 0.1 | 0.2 | $>$ 1h<br>$>$ 30000/80000 | 267.38s<br>18721/55062 | 21.49s<br>1813/3224 |
| $P3$ | 0.1 | 1 | $>$ 1h<br>$>$ 20000/70000 | 142.14s<br>12113/38808 | 5.63s<br>403/970 |
| $WP$ | 0.1 | 1 | 33.48s<br>1683/3692 | 12.27s<br>1521/2859 | 9.95s<br>949/1699 |
| $SB$ | 0.01 | 1 | $>$ 1h<br>$>$ 0/100000 | 15.35s<br>0/4932 | 11.93s<br>0/2743 |
| $CD$ | 0.01 | 1 | 2497.32s<br>2871/28665 | 959.93s<br>9301/26568 | 705.30s<br>1086/13414 |
| $FD$ | 0.1 | 1 | 2638.82s<br>16437/92681 | 440.21s<br>26330/70218 | 273.77s<br>10324/35134 |

Other tests were carried out on tens of similar problems. They showed that the best gains, in running time and number of boxes, of `UCA6*` over `DMBC+` are obtained for problems with low-arity constraints. `UCA6*` remains better than `DMBC+` in running time in the other cases. The experiments also showed that `UCA6-Plus` is always better than `UCA6` in running time and number of boxes. The best gains are obtained when the non-linear constraints contain some nearly axis-parallel sub-regions.

The preliminary experiments are therefore encouraging enough to warrant further investigations and in-depth evaluations.

## 7 Conclusion

Interval-constraint based solvers are usually designed to deliver point-wise solutions. Their techniques are complete and work efficiently for problems with isolated solutions, but might alter both efficiency and compactness of the output representation for many problems with continuum of feasible points. In this paper, we propose a technique for computing inner and outer approximations of numerical CSPs that remedies this state of affairs. The approach works for general non-linear CSPs with equality and inequality constraints. It combines an enhanced splitting policy with the extreme vertex representation of orthogonal polyhedra, as defined in computational geometry. This allows for gains in performance and space requirements. The quality of the output is also enhanced since the inner approximations provide *guaranteed feasible boxes*. In practice, NCSPs with continuum of solutions often occur as sub-problems of higher dimensional NCSPs. In future work, we therefore plan to investigate collaboration strategies between our approximation techniques and standard point-wise interval-based solvers. We will also study alternative implementation schemes purely based on the extreme vertex representation of orthogonal polyhedra (i.e. those do not require intermediate conversion steps to the disjoint box representation).

**Acknowledgments.** Support for this research was provided by the European Commission and the Swiss Federal Education and Science Office (OFES) through the COCONUT project (IST-2000-26063).

## References

1. Aguilera, A.: Orthogonal Polyhedra: Study and Application. PhD thesis, Universitat Politècnica de Catalunya, Barcelona, Spain (1998)
2. Bournez, O., Maler, O., Pnueli, A.: Orthogonal Polyhedra: Representation and Computation. F. Vaandrager and J. Van Schuppen (Eds.), Hybrid Systems: Computation and Control, LNCS 1569 (1999) 46–60 Springer.
3. Bournez, O., Maler, O.: On the Representation of Timed Polyhedra. In: Proceedings of International Colloquium on Automata Languages and Programming (ICALP'2000), Switzerland (2000)
4. Nocedal, J., Wright, S.: Numerical Optimization - Series in Operations Research. Springer (2000)
5. Van Hentenryck, P.: A gentle introduction to Numerica. (1998)
6. ILOG: ILOG Solver. Reference Manual. (2001)
7. Jaulin, L.: Solution Globale et Guarantie de problèmes ensemblistes: application à l'estimation non linéaire et à la commande robuste. PhD thesis, Université Paris-Sud, Orsay (1994)
8. Sam-Haroud, D., Faltings, B.: Consistency techniques for continuous constraints. Constraints **1** (1996) 85–118
9. Garloff, J., Graf, B.: Solving strict polynomial inequalities by Bernstein expansion. In: Symbolic Methods in Control System Analysis and Design. (1999) 339–352
10. Benhamou, F., Goualard, F.: Universally Quantified Interval Constraints. In: Proceedings of the $6^{th}$ International Conference on Principles and Practice of Constraint Programming (CP'2000). (2000) 67–82
11. Lhomme, O.: Consistency techniques for numeric CSPs. In: Proceedings of IJCAI'93. (1993)

12. Collavizza, H., Delobel, F., Rueher, M.: Extending consistent domains of numeric CSP. In: Proceedings IJCAI'99. (1999)
13. Silaghi, M.C., Sam-Haroud, D., Faltings, B.: Search techniques for non-linear CSPs with inequalities. In: Proceedings of the 14th Canadian Conference on AI. (2001)

# Approximation of Relations by Propositional Formulas: Complexity and Semantics

Bruno Zanuttini

GREYC - Université de Caen
bd. Mal Juin, F-14032 Caen Cedex, France
zanutti@info.unicaen.fr

**Abstract.** Selman and Kautz introduced the notion of approximation of a theory and showed its usefulness for knowledge compilation and on-line reasoning. We study here the complexity of the main computational problems related to the approximation of relations (sets of possible worlds) by propositional formulas, and the semantics of reasoning with these approximations (deduction and abduction). The classes of formulas that we consider are those of (reverse-)Horn, bijunctive and affine formulas, which are the most interesting for storing knowledge.
Concerning the computation of approximations, we survey and complete the results that can be found in the literature, trying to adopt a unified point of view. On the contrary, as far as we know this paper is the first real attempt to study the semantics of abduction with the bounds of a theory.

## 1 Introduction

Recently there has been a great amount of research about *knowledge approximation* ([DP92,KKS95,Val95,SK96,KR96] for instance). An approximation of a theory $\Sigma$ representing some knowledge is another theory that approaches it the best (in some precise sense) but allows for efficient reasoning (for instance, deduction) when $\Sigma$ does not. The idea is to store the approximation with the theory itself, and to use it for speeding up further reasoning. Thus computing approximations can be viewed as a *knowledge compilation* task [SK96] ; practically speaking, this compilation is most of the time intended as an *off-line* process performed in order to speed up further *on-line* reasoning.

We are interested here in *propositional* theories. Many papers have focused on them [DP92,KKS95,SK96,Bou98,CS00], may the original theories be given as relations [DP92,KKS95] or as formulas [Val95,SK96,CS00]. In this paper, we consider the case when the original theory $\Sigma$ is given as a relation, which corresponds to the case when konwledge is stored as a set of observations [DP92, ZH02a] (or *possible worlds* [KKS95]) ; in this framework, we study the complexity of computing a *propositional formula* approximating $\Sigma$. The motivation for approximating by formulas is that most of the time formulas are smaller than their set of models, and since wa aim at storing the approximation and reasoning with it, its size is a very important parameter to take into account.

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 242–255, 2002.

Following [SK96], we measure closeness of the approximation to the original theory in terms of sets of models ; we then talk about *Upper Bounds* when the approximation is *logically weaker* than the theoy, and of *Lower Bounds* when it is *logically stronger*. Finally, we are interested in approximations into the classes of (reverse-)Horn, bijunctive and affine formulas, for these classes are the most important ones for reasoning and representing knowledge, in many senses. Indeed, they are, among other properties, tractable for the satisfiability problem, stable by conjunction, by conjunction of facts (unit clauses) and by unit resolution, and tractable for deduction of clauses.

Our contribution is the following. We first survey and complete the results concerning the complexity of computing Least Upper Bounds and Greatest Lower Bounds of relations into these classes of formulas, trying to adopt a unified point of view ; moreover, we give explicit examples and counter-examples, or detail the proofs that can be found in the literature when useful. Secondly, we study the semantics of reasoning with such bounds, i.e., the links between the answers obtained when reasoning with the bounds and those that could have been obtained with the original theory. While deduction has been extensively studied in [Val95,SK96], as far as we know the semantics of abduction have not really been studied in the literature.

The paper is structured as follows. Section 2 is devoted to the explanation of our definitions and conventions. In section 3 we discuss the computation of *upper bounds*, in section 4 that of *lower* bounds, and in section 5 we discuss the semantics of reasoning with these bounds. We end with a summary of the results and further directions for research in section 6.

## 2 Preliminaries

We assume a countable number of propositional variables $x_1, x_2, \ldots$. A *relation* on $\{0,1\}$ is a subset of $\{0,1\}^n$ ; for instance, $R = \{001, 011, 101\}$ is a relation. We view any vector $m$ of $\{0,1\}^n$ as a 0/1 assignment to $n$ variables (in the order of their indices), and $m_i$ denotes the value assigned by $m$ to $x_i$. We write $|R|$ the number of vectors in a relation $R$.

A *literal* is either a variable $x_i$ (*positive* literal) or its negation $\neg x_i$ (*negative* literal). A *clause* is a finite disjunction of literals, and a formula is in *conjunctive normal form* (CNF for short) if it is written as a finite conjunction of clauses. A 0/1 assignment $m$ to the variables of a formula $\phi$ in CNF is a *model* of $\phi$ (written $m \models \phi$) if it satisfies at least one literal in each clause of $\phi$. For instance, the assignment $m = 001$ to the variables $x_1, x_2, x_3$ is a model of the CNF $\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$. An *affine* formula [Sch78,KS98] is a conjunction of linear equations modulo 2. A *model* of an affine formula $\phi$ is an assignment $m$ to its variables that satisfies all the equations (written $m \models \phi$). For instance, $m = 001$ is a model of the affine formula $\phi = (x_1 \oplus x_3 = 1) \wedge (x_1 \oplus x_2 = 0)$. As can be seen, equations play the same role for affine formulas than clauses do for CNFs. If $\phi$ is a CNF or an affine formula, we denote by $\mathcal{M}(\phi)$ the set of its models.

A formula in conjunctive normal form is *Horn* if it has at most one positive literal per clause ; it is reverse-Horn if it has at most one negative literal per clause. For instance, the formula $\phi_1 = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2) \wedge (\neg x_1 \vee \neg x_3)$ is Horn, and $\phi_2 = (x_1 \vee \neg x_2) \wedge (x_3)$ is reverse-Horn. Note that if a CNF is reverse-Horn, then replacing each variable with its negation in it yields a Horn formula, and conversely ; the corresponding models are also strongly linked, in the sense that the models of one are the others' where 0 is replaced with 1 and 1 with 0. For that reason, we only study Horn formulas in the paper ; all the results for reverse-Horn formulas are dual. Finally, a CNF is *bijunctive* if it has at most two literals per clause. For instance, $\phi_1$ above is not bijunctive while $\phi_2$ is. We denote by HORN (resp. REVERSE-HORN, BIJUNCTIVE) the class of all Horn (resp. reverse-Horn, bijunctive) CNFs, and by AFFINE the class of all affine formulas. Unless stated otherwise, when we consider a class of formulas $\mathcal{C}$ it is always one of these. In the paper, the term *formula* groups both CNFs and affine formulas.

Our motivation for studying these classes of formulas comes from their very good computational properties for reasoning. First of all, they are all polynomial for the satisfiability problem : linear for (reverse-)Horn and bijunctive formulas, and quadratic for affine (more precisely, SAT for affine formulas can be solved in time $O(k^2 n)$ if the formula consists of $k$ equations over $n$ variables, with gaussian elimination, see for instance [Cur84]). Secondly, it follows from their definitions that all these classes are stable by conjunction, which allows for instance to update a Horn knowledge base by adding new Horn rules while preserving the Horn form. From their definitions it also follows that they are stable by conjunction of facts (unit clauses) and by unit resolution, which allows for instance one to specify a value for a variable and propagate it while preserving the class, and which also implies that deduction of clauses, one of the most common reasoning problems, is tractable ; indeed, deciding whether $\Sigma$ implies $\ell_1 \vee \cdots \vee \ell_p$ is the same as deciding whether $\Sigma \wedge \overline{\ell_1} \wedge \cdots \wedge \overline{\ell_p}$ is unsatisfiable, and thus stability by conjunction of unit clauses and by unit resolution, in addition of tractability for SAT, makes deduction of clauses tractable. Moreover, in some sense these classes are the only ones to possess tractability for SAT and stability by conjunction, which follows from Schaefer's dichotomy theorem [Sch78]. Finally, these classes allow efficient restricted forms of abduction [Val00,Z02b] while this problem, also of major importance for reasoning, is $\Sigma_2^P$-complete in general [EG95]. For all these reasons we argue that these four classes are the most interesting ones for reasoning and representing propositional knowledge.

A formula $\phi$ is said to *describe* a relation $R$ if $R = \mathcal{M}(\phi)$ ; $\phi$ is called a *description* of $R$. For $m, m' \in \{0,1\}^n$, let us write $m \wedge m'$ the vector $m'' \in \{0,1\}^n$ such that $\forall i = 1, \ldots, n, m''_i = m_i \wedge m'_i$, and similarly for $\vee$ and $\oplus$. The following result, proved by Schaefer, is essential for the paper.

**Proposition 1 ([Sch78]).** *Let $R$ be a relation. Then $R$ has at least one description in*
*(i)* HORN *if and only if* $\forall m, m' \in R, m \wedge m' \in R$,
*(ii)* BIJUNCTIVE *iff* $\forall m, m', m'' \in R, (m \vee m') \wedge (m \vee m'') \wedge (m' \vee m'') \in R$,
*(iii)* AFFINE *iff* $\forall m, m', m'' \in R, m \oplus m' \oplus m'' \in R$.

## 3 Upper Bounds

We are first interested in the computation of *upper bounds* of relations. The idea is that we weaken the original theory in order to obtain a more interesting representation.

**Definition 1 ($\mathcal{C}$-upper bound).** *Let $R \subseteq \{0,1\}^n$ be a relation and $\mathcal{C}$ a class of formulas. A formula $\phi$ on the variables $x_1, \ldots, x_n$ is called a* $\mathcal{C}$-upper bound *($\mathcal{C}$-UB) of $R$ if $\phi \in \mathcal{C}$ and $R \subseteq \mathcal{M}(\phi)$. If this holds for no formula $\phi'$ with $\mathcal{M}(\phi') \subset \mathcal{M}(\phi)$, $\phi$ is called a* $\mathcal{C}$-least upper bound *($\mathcal{C}$-LUB) of $R$.*

Of course, the most interesting upper bounds are the *least* ones, for they approach the original theory the closest and thus yield a smaller loss of information than general upper bounds. Proposition 1 immediately yields the following result for $\mathcal{C}$-LUBs.

**Proposition 2.** *Assume $\mathcal{C}$ is either* Horn, Bijunctive *or* Affine. *Then the $\mathcal{C}$-LUB of any relation is unique up to logical equivalence.*

*Proof.* From Proposition 1 it follows that the set of models of the $\mathcal{C}$-LUB of a relation $R$ is the closure of $R$ under the operation corresponding to the class. □

*Example 1.* Consider the relation $R = \{011, 110, 101\}$. It is easily seen that $011 \wedge 110 = 010$, $011 \wedge 101 = 001$, $110 \wedge 101 = 100$ and $010 \wedge 001 = 011 \wedge 100 = \cdots = 000$, and finally that the set of models of any Horn-LUB of $R$ is $\{0,1\}^3 \backslash \{111\}$.

For sake of simplicity now, we will say a formula $\phi$ is *the* $\mathcal{C}$-LUB of a relation. Our problem is to compute such a $\phi$ when $\mathcal{C}$ is one of the classes of interest here.

We first consider Horn-LUBs. Unfortunately, there cannot be an input-polynomial algorithm for computing the Horn-LUB of a relation, since the former can be exponentially bigger than the latter, as shown in [KKS95] (where an explicit example is given).

**Proposition 3 ([KKS95, Theorem 6]).** *There exist relations on $2n$ variables, consisting of $3n(n-1)$ vectors but whose* Horn-*LUB has $2^n$ clauses.*

Since there cannot be an input-polynomial algorithm for this problem, it is worth wondering whether there exist an *output-polynomial* one [JYP88]. But it is shown in [KPS93,Kha95] that this problem is harder than the enumeration of all the minimal transversals of an hypergraph, which is open and for which the best known algorithm is subexponential [Kha95].

Now we turn our attention to bijunctive and affine formulas. Contrary to the Horn case, for both classes polynomial algorithms are given in the literature. The idea for bijunctive formulas is that we only need to project the relation onto every pair of variables and compute the associated clauses, and the proof for affine formulas uses a strong link between the sets of models of affine formulas and vector spaces, which allows to use the notion of *basis* of such a space.

**Proposition 4 ([DP92, Lemma 3.18]).** *The* Bijunctive-*LUB of a relation $R \in \{0,1\}^n$ can be computed in time $O(|R|n^2)$.*

**Proposition 5 ([Z02a, Proposition 6]).** *The* Affine-*LUB of a relation $R \in \{0,1\}^n$ can be computed in time $O(|R|n^3 + n^4)$.*

## 4 Lower Bounds

We are now interested in *lower bounds* of relations, i.e., in *strengthening* the original theory. The following definition is dual to Definition 1.

**Definition 2 ($\mathcal{C}$-lower bound).** *Let $R \subseteq \{0,1\}^n$ be a relation and $\mathcal{C}$ a class of formulas. A formula $\phi$ on the variables $x_1, \ldots, x_n$ is called a* $\mathcal{C}$-lower bound *($\mathcal{C}$-LB) of $R$ if $\phi \in \mathcal{C}$ and $\mathcal{M}(\phi) \subseteq R$. If this holds for no formula $\phi'$ with $\mathcal{M}(\phi) \subset \mathcal{M}(\phi')$, $\phi$ is called a* $\mathcal{C}$-greatest lower bound *($\mathcal{C}$-GLB) of $R$.*

Dually to the case of upper bounds we are mainly interested in *greatest* lower bounds. But contrary to its $\mathcal{C}$-LUB, the $\mathcal{C}$-GLB of a relation is not unique in general. We are thus interested in two problems : find one $\mathcal{C}$-GLB of a given theory, and find one with the maximum number of models. Indeed, two $\mathcal{C}$-GLBs of the same relation have not even the same number of models in the general case, and one of them may even be exponentially bigger than another, as the next example illustrates it.

*Example 2.* Consider the relation over the variables $x_1, \ldots, x_n$ :

$$R = \{m100 | m \in \{0,1\}^{n-3}\} \cup \{0 \ldots 0010, 0 \ldots 0001\}$$

where $m100$ stands for the assignment of $m_i$ to $x_i$ for $i = 1, \ldots, n-3$, 1 to $x_{n-2}$ and 0 to $x_{n-1}, x_n$, $0 \ldots 0010$ stands for the assignment of 0 to $x_1, \ldots, x_{n-3}$, 0 to $x_{n-2}$, 1 to $x_{n-1}$ and 0 to $x_n$, and similarly for $0 \ldots 0001$. It is easily seen with Proposition 1 that $S_{ha} = \{m100 | m \in \{0,1\}^{n-3}\}$ is the set of models of a Horn- and Affine-GLB of $R$, and $S_b = \{m100 | m \in \{0,1\}^{n-3} \cup \{0 \ldots 0010\}$ that of a Bijunctive-GLB of it. On the other hand, $S'_h = \{0 \ldots 0010\}$ is also the set of models of a Horn-GLB of $R$, and $S'_{ba} = \{0 \ldots 0010, 0 \ldots 0001\}$ of a Bijunctive- and Affine-GLB. However, for all $n$ it holds that $|R| = 2^{n-3} + 2$, $|S_{ha}| = 2^{n-3}$, $|S_b| = 2^{n-3} + 1$ and $|S'_h| = 1$, $|S'_{ba}| = 2$.

### 4.1 Computing One GLB

The first natural problem concerning the computation of GLBs is the one of computing *one* GLB. For the classes we are interested in the problem appears to be polynomial, mainly because one can compute in polynomial time a formula $\phi \in \mathcal{C}$ given the set of its models [DP92,KS98,ZH02a]. Thus we can first compute the set of models $S$ of a $\mathcal{C}$-GLB of $R$, which can be done by selecting vectors from $R$ as long as the closure of the current $S$ is included in $R$, and then a description of $S$. Since the size of $S$ is by definition less than the size of $R$, the whole process is polynomial.

**Proposition 6 (see also [KPS93, Theorem 2]).** *A Horn-GLB of a relation $R \subseteq \{0,1\}^n$ can be computed in time $O(|R|^2 n^2)$.*

*Proof.* We begin by computing the set of models $S$ of the Horn-GLB. Initialize $S$ to $\emptyset$, and perform the following until there is no more convenient vector $m$ in $R$ : pick $m \in R$ such that $\forall m' \in S, m \wedge m' \in R$ and set $S \leftarrow S \cup \{m\} \cup \{m \wedge m' | m' \in S\}$. Then $S$ is always closed under $\wedge$ and included in $R$, and when no more

convenient $m$ can be chosen in $R$, then $S$ is obviously maximal for set inclusion. Then compute a formula $\phi$ describing $S$.

The running time is established as follows : since at most $|R|$ vectors can be added to $S$, and at each step one has to check that at most $|R|$ products $m \wedge m'$ are in $R$ (which requires $O(n)$ steps if $R$ is initially sorted into a decision tree, in time $O(|R|n)$), computing $S$ requires time $O(|R|^2 n)$. Then $|S| \leq |R|$ holds, and describing $S$ into a Horn formula requires $O(|S|^2 n^2)$ steps [DP92,ZH02a]. □

**Proposition 7.** *A* Bijunctive- *or* Affine-*GLB of a relation* $R \in \{0,1\}^n$ *can be computed in time* $O(|R|^3 n + |R|^2 n^2)$.

*Proof.* The proof is the same as for Proposition 6, except that the test for stability requires three vectors ; thus computing $S$ requires time $O(|R|^3 n)$ instead of $O(|R|^2 n)$, and describing $S$ still requires $O(|S|^2 n^2)$ steps [ZH02a]. □

However, computing one GLB may turn out to be rather uninteresting ; indeed, as shown in Example 2 one can find a GLB $\phi$ whose set of models is exponentially smaller than that of another GLB $\phi'$. In such a case, it is natural to consider that the bounds with exponentially many models are the best ones. For this reason, the problem of computing a GLB *with the maximum number of models over all GLBs* is more interesting.

### 4.2 Computing a GLB with the Maximum Number of Models

We now consider the problem of computing a GLB that has the maximum number of models over all GLBs. More formally, for $R$ a relation and $\mathcal{C}$ a class of formulas, let us call a $\mathcal{C}$*-maxGLB of* $R$ any $\mathcal{C}$-GLB $\phi$ of $R$ such that no other $\mathcal{C}$-GLB $\phi'$ of $R$ has more models than $\phi$ ; we emphasize that we do not consider here the relations of inclusion between the sets of models of $\phi$ and $\phi'$, but only the number of these models.

As for $\mathcal{C}$-LUBs, the results here are not the same for all the classes we are interested in. In each case computing a $\mathcal{C}$-maxGLB is hard, but it is NP-hard for Horn and Bijunctive while subexponential, and thus unlikely to be NP-hard [SH90], for Affine.

In order to show NP-hardness, we show that it is NP-hard to compute the *set of models* of a maxGLB ; indeed, the models of any Horn or bijunctive formula can be enumerated in output-polynomial time, and since a GLB of $R$ has by definition less than $|R|$ models, we deduce that up to a polynomial overcost computing the set of models of a maxGLB $\phi$ is not harder than computing $\phi$ ; thus if the former computation is NP-hard, then the latter also is.

**Proposition 8 (see also [KPS93, Theorem 2]).** *If* $P \neq NP$, *a* Horn-*maxGLB of a relation* $R \subseteq \{0,1\}^n$ *cannot be computed in time polynomial in* $|R|$ *and* $n$.

*Proof.* We consider the associated decision problem : given $R$ and $k \leq |R|$, is there a Horn-glb of $R$ that consists in at least $k$ vectors ?, and we reduce the problem `INDEPENDENT-SET`, which is known to be NP-complete (see [GJ83]), to

this one. Let us recall that an independent set of a graph $G$ is a subset $V'$ of its vertices such that no two vertices in $V'$ are joined by an edge of $G$. The problem INDEPENDENT-SET is the following :

*Input :* A graph $G = (V, E)$ and a positive integer $k \leq |V|$

*Question :* Does $G$ contain an independent set of size at least $k$ ?

The reduction is the following : first consider $|E|$ variables $x_1, \ldots, x_{|E|}$, each variable $x_i$ corresponding to the edge $e_i$ of $G$, and associate to each vertex $v_j$ of $G$ a vector $m^{[j]}$ such that for each variable $x_i$, $m_i^{[j]}$ is 1 if and only if $v_j \in e_i$. Then introduce $|V|$ new variables $x_{|E|+1}, \ldots, x_{|E|+|V|}$, each variable $x_{|E|+k}$ corresponding to the vertex $v_k$ of $G$, and complete each vector $m^{[j]}$ with $m^{[j]}_{|E|+k} = 1$ if and only if $j = k$, i.e., if and only if $m^{[j]}$ is the vector associated to $v_k$ (this distinguishes, for example, the two vertices of the connected graph with only one edge). Finally, add the all-0 vector $\mathbf{0}$. Call $R$ this relation ; it clearly contains $|V| + 1$ vectors and $|V| + |E|$ variables, and its construction from $G$ is polynomial.

We show that $G$ has an independent set of size $k$ if and only if $R$ has a HORN-glb of size at least $k + 1$. Assume first that $G$ has an independent set $V' = \{v_1, v_2, \ldots, v_k\}$ (without loss of generality). Consider the set of vectors $S = \{\mathbf{0}, m^{[1]}, \ldots, m^{[k]}\} \subseteq R$. Obviously, for all $i$, $\mathbf{0} \wedge m^{[i]} = \mathbf{0} \in S$. Now for $m^{[i]}, m^{[j]} \in S, m^{[i]} \neq m^{[j]}$, since $V'$ is an independent set there is no edge joining $v_i$ and $v_j$, thus $m^{[i]}$ and $m^{[j]}$ have no 1 in common on the first $|E|$ variables ; since $m^{[i]} \neq m^{[j]}$, they have no 1 in common on the last $|V|$ variables neither, and finally $m^{[i]} \wedge m^{[j]} = \mathbf{0} \in S$. We conclude that $S$ is stable by $\wedge$, thus $R$ has a HORN-GLB of size at least $k + 1$.

Conversely, assume that $G$ has no independent set of size $k$. This means that for any subset $V' \subseteq V, |V'| = k$, there exist $v_i, v_j \in V'$ joined by an edge $e$. We conclude that for any subset $S$ of $R$ of size at least $k+1$, there exist $m^{[i]}, m^{[j]} \in S$ with one 1 in common on the first $|E|$ variables, and since $m^{[i]} \neq m^{[j]}$ none in common among the last $|V|$ variables. Thus $m^{[i]} \wedge m^{[j]}$ is all-0 except for at least one of the first $|E|$ variables. Since every vector in $R$ is either all-0, or 1 on exactly one of the last $|V|$ variables, $m^{[i]} \wedge m^{[j]} \notin R$. Thus no subset of $R$ is both of size at least $k + 1$ and stable by $\wedge$, thus $R$ has no HORN-GLB of size $k + 1$ or more. □

**Proposition 9.** *If $P \neq NP$, a* BIJUNCTIVE-*maxGLB of a relation $R \subseteq \{0,1\}^n$ cannot be computed in time polynomial in $|R|$ and $n$.*

*Proof.* As for Proposition 8, we reduce the problem INDEPENDENT-SET to the associated decision problem. We encode a given instance of INDEPENDENT-SET in the same manner as for Horn, except that we replace 0 with 1 and 1 with 0 for each of the first $|E|$ variables. We let the last $|V|$ variables distinguish all the vectors as before, with exactly one 1 per vector. The vector $\mathbf{0}$ is similarly replaced with $\mathbf{A} = 1 \ldots 10 \ldots 0$.

Assume $G$ has an independent set $\{v_1, v_2, \ldots, v_k\}$, and consider the subset $S = \{\mathbf{A}, m^{[1]}, \ldots, m^{[k]}\}$ of $R$. We show that for any $m, m', m'' \in S$, $m_0 = (m \vee m') \wedge (m \vee m'') \wedge (m' \vee m'') \in S$. Indeed, $m$ and $m'$ have no 0 in common on the first

$|E|$ variables, for either the associated vertices have no edge in common or one of the two vectors is all-1 on these variables. Thus $m \vee m'$ is 1 on each one of the first $|E|$ variables. The same holds for $m \vee m''$ and $m' \vee m''$, thus $m_0$ assigns 1 to each one of the first $|E|$ variables. It is easily seen that it assigns 0 to the last $|V|$ variables whatever $m$, $m'$ and $m''$ may be, thus $m_0 = \mathbf{A} \in S$. Thus $R$ has a BIJUNCTIVE-GLB of size at least $k+1$. The converse is easily seen to be true, for if $m$ and $m'$ (without loss of generality) have one edge in common, then $m \vee m'$ assigns 0 to the corresponding variable, thus also $m_0$, but $m_0$ still assigns 0 to all of the last $|V|$ variables ; since such a vector does not exist in $R$, the associated subset of $R$ cannot be completed into a BIJUNCTIVE-LB of $R$. □

On the contrary, a subexponential algorithm is given in [Z02a] for the affine case. As for computing an AFFINE-GLB, it uses the correspondence with vector spaces and the notion of basis. Note that subexponential algorithms may be reasonable in practice, while NP-hard problems are unlikely to be tractable. Note also that subexponential algorithms are unlikely to be NP-hard [SH90].

**Proposition 10 ([Z02a, Proposition 7]).** *An* AFFINE-*maxGLB of a relation* $R \subseteq \{0,1\}^n$ *can be computed in time* $O(|R|n(\log\log|R|)2^{(\log|R|)^2})$.

## 5 Reasoning with Bounds

Finally, we study the semantics of reasoning with the bounds of a theory. The problem is to characterize the results obtained when reasoning with the bounds in function of those obtained when reasoning with the theory itself. Of course, such characterizations are of major importance for being able to use the bounds of the theory for speeding up further answering to queries. Indeed, the idea is to use as often as possible *only* the approximation for answering the queries that are asked to the original theory, for it is meant to have much better computational properties for reasoning ; that is why we must be able to deduce the right answer (the one that the original theory would have given) from the one that we compute with the approximation. Moreover, if it appears that the bounds allow to solve any reasoning task the theory will be possibly sollicited for, we may even store only these bounds and forget the theory itself.

We however emphasize that the size of a bound must be comparable to or lower than the size of the original theory, for otherwise, on one hand there would be no gain in reasoning with it, even if reasoning is more efficient in the class of the approximation than in the general case, and on the other hand its storage could need too much memory.

### 5.1 Deduction

We talk about deduction when we want to answer the question "does $\Sigma$ logically implies $\alpha$ ?", where $\Sigma$ is a theory and $\alpha$ (the *query*) is another theory ; the answer is positive (written $\Sigma \models \alpha$) if and only if every model of $\Sigma$ is a model of $\alpha$, i.e., $\alpha$ is true in all the situations where $\Sigma$ is. Deduction is in general coNP-complete,

since $\Sigma$ implies $\alpha$ if and only if $\Sigma \wedge \overline{\alpha}$ is unsatisfiable, and thus it is worth considering the semantics of deduction with bounds in classes of formulas for which it becomes tractable.

Selman and Kautz [SK96] have studied the semantics of deduction with bounds. The following result is given in [SK96] for Horn bounds, but it can be straightforwardly extended to any class of formulas.

**Proposition 11 ([SK96, Section 2.2]).** *Let $\Sigma$ and $\alpha$ be two propositional theories, and let $\mathcal{C}$ be a class of propositional formulas. Let $\Sigma_{LUB}$ be a $\mathcal{C}$-LUB of $\Sigma$, and $\Sigma_{GLB}$ a $\mathcal{C}$-GLB. Then :*
*(i) If $\Sigma_{LUB}$ implies $\alpha$, then $\Sigma$ implies $\alpha$*
*(ii) If $\Sigma_{GLB}$ does not imply $\alpha$, then $\Sigma$ does not imply $\alpha$.*

It follows that we can use the bounds for speeding up reasoning with the original theory in the following manner. When asked whether $\Sigma \models \alpha$, we first decide whether $\Sigma_{LUB} \models \alpha$ ; if the bound $\Sigma_{LUB}$ is taken in a class of formulas for which deduction is tractable, and if the size of $\Sigma_{LUB}$ is comparable to or less than the size of $\Sigma$, then this test is efficient. If its answer is positive, then we can conclude. Otherwise, we decide whether $\Sigma_{GLB} \models \alpha$ ; if the answer is negative, once again we can conclude. In case it is positive, then we have no other solution than to decide directly $\Sigma \models \alpha$, but we have only lost a small amount of time. For more details and examples we refer the reader to [SK96] for instance.

When restricting the form of the queries, the $\mathcal{C}$-LUB of a relation can even allow to solve *exactly* deduction tasks without any use of the original theory, as stated in the next proposition (whose proof can be straightforwardly adapted from del Val's).

**Proposition 12 (see [Val95, Theorem 1]).** *Assume $\mathcal{C}$ is either* Horn, Bijunctive *or* Affine. *Let $\Sigma$ be a theory and $\alpha \in \mathcal{C}$. Then the $\mathcal{C}$-LUB of $\Sigma$ implies $\alpha$ if and only if $\Sigma$ itself implies $\alpha$.*

### 5.2 Abduction

We now consider the problem of *abduction*. Contrary to the case of deduction, as far as we know the semantics of this process with bounds have not really been studied in the literature, except in a quite special case in [KKS95, Section 5]. This section is a first step into this direction.

Abduction consists in searching for *explanations* of observations, knowing a background theory. More formally, a background theory is simply a propositional theory $\Sigma$, supposed to be satisfiable, and an observation is another theory $\alpha$. We explicit what an explanation is in the following definition.

**Definition 3 (explanation).** *Let $\Sigma$ and $\alpha$ be two propositional theories, and $H$ a subset of $Var(\Sigma) \backslash Var(\alpha)$ (the* set of hypotheses*). Then an* explanation *of $\alpha$ knowing $\Sigma$ over $H$ is a conjunction $E$ of literals such that :*
*(i) $Var(E) \subseteq H$*
*(ii) $\Sigma \wedge E$ implies $\alpha$*
*(iii) $\Sigma \wedge E$ is satisfiable.*
*If in addition there is no proper subset of $E$ with the same properties, $E$ is called a* minimal *explanation of $\alpha$ knowing $\Sigma$ over $H$.*

*Example 3.* Let $\Sigma$ be the theory represented by the CNF formula $(x_1 \vee x_3 \vee \neg x_5) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee \neg x_3 \vee x_6) \wedge (\neg x_4 \vee x_5)$, let $\alpha$ be the clause $(x_5 \vee x_6)$ and $H$ the set of hypotheses $\{x_1, x_2, x_3\}$. Then $E = \{x_1, \neg x_2\}$ is an explanation of $\alpha$ knowing $\Sigma$ over $H$. Moreover, it is a *minimal* explanation, since neither $E' = \{x_1\}$ nor $E'' = \{\neg x_2\}$ are explanations.

With this definition, the task of abduction consists in finding, given $\Sigma$, $\alpha$ and $H$, a minimal explanation $E$ of $\alpha$ knowing $\Sigma$ over $H$. It is well-known that in general abduction is $\Sigma_2^P$-complete [EG95], and NP-hard if we restrict $\Sigma$ to be a Horn formula [EG95]. On the other hand, for bijunctive and affine formulas it becomes polynomial if in addition we impose some restrictions to the form of $\alpha$ (see for instance [Val00,Z02b]). Therefore, in all cases it is interesting to study the semantics of abduction with bounds.

We first give general remarks that help understanding these semantics in the general case, independently of the class of the approximation and of the restrictions of the general problem.

**Proposition 13.** *Let $\Sigma$ and $\alpha$ be two propositional theories, and $H$ a subset of $Var(\Sigma) \backslash Var(\alpha)$. Let $\mathcal{C}$ be a class of formulas, and $\Sigma_{LUB}$ (resp. $\Sigma_{GLB}$) be a $\mathcal{C}$-LUB (resp. a $\mathcal{C}$-GLB) of $\Sigma$. Let $E$ be a conjunction of literals formed upon the variables in $H$. Then :*
*(i) If $\Sigma_{GLB} \wedge E$ is satisfiable, then $\Sigma \wedge E$ is satisfiable*
*(ii) If $\Sigma_{GLB} \wedge E$ does not imply $\alpha$, then $\Sigma \wedge E$ does not imply $\alpha$*
*(iii) If $\Sigma_{LUB} \wedge E$ is not satisfiable, then $\Sigma \wedge E$ is not satisfiable*
*(iv) If $\Sigma_{LUB} \wedge E$ implies $\alpha$, then $\Sigma \wedge E$ implies $\alpha$.*

*Proof.* Points (ii) and (iv) are stated in Proposition 11, and points (i) and (iii) directly follow from the definition of the bounds. □

Now we study more precise classes of formulas. When restricting the form of the observations and explanations, it sometimes becomes possible to perform abduction with the $\mathcal{C}$-LUB of a theory. In these cases, it thus becomes possible to use the approximations of a theory for performing abduction in the same manner as we can use them for performing deduction.

We first consider the case of HORN-LUBs. We show that if the query is *positive*, i.e., is logically equivalent to at least one conjunction of positive literals, the *positive* explanations that can be found with $\Sigma$ are exactly the same as those that can be found with its HORN -LUB. The point is that when the background knowledge is Horn, a positive observation can be explained only by a conjunction of positive literals, as shown in the next lemma.

**Lemma 1 (generalization of [RK87, Corollary 4]).** *Let $\Sigma$ be a Horn theory, $\alpha$ a positive formula and $E$ a minimal explanation of $\alpha$ knowing $\Sigma$. Then $E$ contains only positive literals.*

*Proof.* Assume, for sake of contradiction, that $E$ contains $\neg x_i$ for a certain $x_i \in Var(\Sigma) \backslash Var(\alpha)$. Then there is a model $m$ of $\Sigma$ with $m \models \alpha$ and $m \models E$, thus in particular $m_i = 0$. We show that $\Sigma \wedge E \backslash \{\neg x_i\}$ implies $\alpha$, and thus that $E$ is not minimal ; for that purpose, we show that there cannot be a model $m'$ of $\Sigma$ with $m' \not\models \alpha$ and $m' \models E \backslash \{\neg x_i\}$. Indeed, if there is such an $m'$, then $m'' = m \wedge m'$

must satisfy $\Sigma$ by Proposition 1. But by construction, $m''$ satisfies $E$, since $m$ and $m'$ are equal over the variables in $E\backslash\{\neg x_i\}$ and $m_i = 0$, thus $m''_i = 0$ ; but $m''$ does not satisfy $\alpha$, since $m' \not\models \alpha$, $m''$ assigns 0 to all the variables of $\alpha$ to which $m'$ assigns 0, and $\alpha$ is positive. Thus $m''$ contradicts the fact that $\Sigma \wedge E$ implies $\alpha$. □

Using Lemma 1, we can then show that performing abduction with the HORN-LUB of $\Sigma$ is the same as with $\Sigma$ itself when considering only positive observations and explanations.

**Proposition 14.** *Let $\Sigma$ be a theory, $\alpha$ a positive observation, and $H$ a subset of $Var(\Sigma)\backslash Var(\alpha)$. Let $\Sigma_{LUB}$ be the* HORN-*LUB of $\Sigma$. If $E$ is a minimal explanation of $\alpha$ knowing $\Sigma_{LUB}$ over $H$, then $E$ is a minimal explanation of $\alpha$ knowing $\Sigma$ over $H$. Conversely, if $E$ is a minimal* positive *explanation of $\alpha$ knowing $\Sigma$ over $H$, then $E$ is a minimal explanation of $\alpha$ knowing $\Sigma_{LUB}$ over $H$.*

*Proof.* Let $E$ be a minimal explanation of $\alpha$ knowing $\Sigma_{LUB}$. We know by Proposition 13 that $\Sigma \wedge E$ implies $\alpha$. Since $\Sigma_{LUB} \wedge E$ is satisfiable, there is a model $m$ of $\Sigma_{LUB}$ with $m \models E$, thus there is a model $m'$ of $\Sigma$ with $m'$ bitwise greater than or equal to $m$ (by the construction of Proposition 2) ; since $E$ is positive (Lemma 1), $m'$ also satisfies $E$, thus $\Sigma \wedge E$ is satisfiable. There is only left to show that $E$ is minimal knowing $\Sigma$. If it were not the case, then there would be a $E' \subset E$ with $\Sigma \wedge E'$ satisfiable, thus $\Sigma_{LUB} \wedge E'$ satisfiable by Proposition 13, and such that $\Sigma \wedge E'$ would imply $\alpha$ but $\Sigma_{LUB} \wedge E'$ would not (for otherwise $E$ would not be minimal knowing $\Sigma_{LUB}$). Thus there would be a model $m$ of $\Sigma_{LUB}$ satisfying $E'$ but not $\alpha$, thus a model $m'$ of $\Sigma$ satisfying $E'$ but not $\alpha$ (again by the construction of Proposition 2) ; thus $\Sigma \wedge E'$ would not imply $\alpha$, contradiction.

Conversely, let $E$ be a minimal positive explanation of $\alpha$ knowing $\Sigma$. We then know by Proposition 13 that $\Sigma_{LUB} \wedge E$ is satisfiable. We also know that $\Sigma_{LUB} \wedge E$ implies $\alpha$ by the same proof as with $E'$ in the previous paragraph. Finally, by minimality of $E$ for $\Sigma$ the first part of the Proposition shows that there cannot be a $E' \subset E$ with $E'$ an explanation knowing $\Sigma_{LUB}$, thus that $E$ is minimal knowing $\Sigma_{LUB}$. □

Now we consider the case of BIJUNCTIVE-LUBs. Parallel to the restriction to *positive* observations for HORN-LUBs, we must now restrict to observations consisting of one clause of one or two literals. Now parallel to Lemma 1, we show in the next lemma that a minimal explanation of such an observation knowing a bijunctive theory must contain at most one literal.

**Lemma 2.** *Let $\Sigma$ be a bijunctive theory, $\alpha$ a clause of one or two literals and $E$ a minimal explanation of $\alpha$ knowing $\Sigma$. Then $E$ contains zero or one literal.*

*Proof.* Since $E$ is an explanation of $\alpha$ knowing $\Sigma$, it holds that $\Sigma$ implies $C = (E \rightarrow \alpha)$ ; writing $E = \{\ell_{i_1}, \ldots, \ell_{i_p}\}$ (maybe empty) and $\alpha = \ell_{j_1} \vee \ell_{j_2}$ (the case $\alpha = \ell_{j_1}$ is similar), we get $C = (\ell_{i_1} \wedge \ldots \wedge \ell_{i_p} \rightarrow \ell_{j_1} \vee \ell_{j_2})$, i.e., $C = (\overline{\ell_{i_1}} \vee \cdots \vee \overline{\ell_{i_p}} \vee \ell_{j_1} \vee \ell_{j_2})$. Since $\Sigma$ is bijunctive, there is a subclause $C'$ of $C$ consisting of two literals that is implied by $\Sigma$ ; thus either (i) $C'$ is a $\overline{\ell_{i_a}} \vee \overline{\ell_{i_b}}$, but this implies $\Sigma \rightarrow \overline{E}$, which contradicts the fact that $\Sigma \wedge E$ is satisfiable, or

(ii) $C'$ is $\ell_{j_1} \vee \ell_{j_2}$, but this means $\Sigma \rightarrow \alpha$ and thus $E$, being minimal, is empty, or finally (iii) $C'$ is a $\overline{\ell_{i_a}} \vee \ell_{j_b}$, and thus $\Sigma$ implies $\ell_{i_a} \rightarrow \ell_{j_b}$, which in turn implies $\ell_{i_a} \rightarrow \alpha$, and thus $E = \emptyset$ or $E = \{\ell_{i_a}\}$ is a minimal explanation. □

Using Lemma 2, we can now show that, parallel to Proposition 14, performing abduction with the BIJUNCTIVE-LUB of $\Sigma$ is the same as with $\Sigma$ itself when considering only clauses of one or two literals as observations and explanations containing zero or one literal.

**Proposition 15.** *Let $\Sigma$ be a theory, $\alpha$ a clause of one or two literals, and $H$ a subset of $Var(\Sigma) \backslash Var(\alpha)$. Let $\Sigma_{LUB}$ be the* BIJUNCTIVE-*LUB of $\Sigma$. If $E$ is a minimal explanation of $\alpha$ knowing $\Sigma_{LUB}$ over $H$, then $E$ is a minimal explanation of $\alpha$ knowing $\Sigma$ over $H$. Conversely, if $E$ is a minimal explanation containing one or zero literal of $\alpha$ knowing $\Sigma$ over $H$, then $E$ is a minimal explanation of $\alpha$ knowing $\Sigma_{LUB}$ over $H$.*

*Proof.* Let $E$ be a minimal explanation of $\alpha$ knowing $\Sigma_{LUB}$ over $H$. Proposition 13 tells us that $\Sigma \wedge E$ implies $\alpha$. Now by Lemma 2 we know that $E$ contains at most one literal ; since $\Sigma$ is satisfiable, if $E = \emptyset$ then $\Sigma \wedge E$ is satisfiable ; now if $E = \{\ell_{i_a}\}$, then the clause $(\overline{\ell_{i_a}})$ is not implied by $\Sigma_{LUB}$ (since $\Sigma_{LUB} \wedge E$ is satisfiable) ; but $\Sigma_{LUB}$ is logically equivalent to the conjunction of all the clauses of length 2 or less that are implied by $\Sigma$, thus $\Sigma$ does not imply $(\overline{\ell_{i_a}})$ and we deduce that $\Sigma \wedge (\ell_{i_a})$ is satisfiable. Finally, if $E$ is not minimal for $\Sigma$ then the only possibility is that $E$ contains one literal and $\emptyset$ is an explanation of $\alpha$ knowing $\Sigma$ ; but this means that $\Sigma$ implies $\alpha$, and since $\alpha$ contains at most two literals we deduce that $\Sigma_{LUB}$ implies $\alpha$, i.e., that $E$ is not minimal for $\Sigma_{LUB}$.

The converse is established as for Proposition 14. □

Finally, we consider aboduction with AFFINE-LUBs. Unfortunately, in the framework for abduction defined here, no link other than those exhibited in Proposition 13 seems to exist between the explanations found with this LUB and those that can be found with the theory itself, even if we restrict ourselves to observations consisting of one variable and to explanations containing at most two literals. The next example illustrates that fact.

*Example 4.* Consider $\Sigma = \{001, 010, 100\}$ over the variables $x_1, x_2, x_3$. It is easily seen that the AFFINE-LUB of $\Sigma$ is $(x_1 \oplus x_2 \oplus x_3 = 1)$. Now consider the observation $\alpha = x_3$ and the set of hypotheses $H = \{x_1, x_2\}$. Then $E = x_1 \wedge x_2$ is a minimal explanation of $\alpha$ knowing the AFFINE-LUB of $\Sigma$ over $H$ ; but $\Sigma \wedge E$ is not satisfiable.

The intuition behind this fact is that the relations between variables that are preserved between a theory and its AFFINE-LUB are the relations that can be expressed by a *linear equation* ; but explanations as defined here are meant to be *conjunctions* of literals, and this is intuitively why they do not behave well. We could define explanations in a more general form (for instance as any propositional formula over $H$), but this is out of scope here.

**Table 1.** Summary of results for computing approximations

| **Computing...** | (Reverse-)Horn | Bijunctive | Affine |
|---|---|---|---|
| the $\mathcal{C}$-LUB | exp output | $\|R\|n^2$ | $\|R\|n^3 + n^4$ |
| one $\mathcal{C}$-GLB | $\|R\|^2n^2$ | $\|R\|^3n + \|R\|^2n^2$ | $\|R\|^3n + \|R\|^2n^2$ |
| a $\mathcal{C}$-maxGLB | NP-hard | NP-hard | subexponential |

## 6 Conclusion

We have settled the complexity of the main problems concerning the computation of Least Upper Bounds and Greatest Lower Bounds of relations into the main classes of formulas that allow efficient storage of knowledge and reasoning, namely the classes of Horn, reverse-Horn, bijunctive and affine formulas. The results are summarized in Table 1. It appears that computing one GLB is polynomial, while computing the LUB is polynomial only for the classes of bijunctive and affine formulas, and computing a GLB with the maximum number of models is NP-hard for Horn and bijunctive formulas while subexponential for affine.

Beside this, we have studied the semantics of reasoning with the bounds of a theory. As far as we know, the semantics of abduction with LUBs had not really been studied before. We have shown that assuming restrictions about the form of the observation and of the explanations, we can perform abduction with the Horn- or Bijunctive-LUB of a theory and get the same explanations as if we had reasoned with the theory itself. These results show that assuming these restrictions, abduction can be performed with the bounds of a theory in the same manner as deduction.

This study of the semantics of abduction is of course incomplete, and this problem would certainly benefit further exploration. Also, an interesting problem is the one of computing Horn-renamable approximations of relations and studying their semantics ; Horn-renamable formulas are those formulas that we can transform into a Horn one by replacing some variables with their negation everywhere in the formula. This class includes Horn and reverse-Horn formulas as well as satisfiable bijunctive formulas, and is tractable for satisfiability and deduction as well. Thus although it is not stable by conjunction, which makes it less interesting for storing knowledge as a set of rules, it is certainly worth studying the complexity of computing approximations of relations into this class. As far as we know, the only study about Horn-renamable approximations can be found in [Bou98].

## References

[Bou98] Boufkhad, Y., Algorithms for propositional KB approximation, in : *Proc. 15th National Conference on Artificial Intelligence (AAAI'98)* (Madison, USA), Menlo Park : AAAI Press / MIT Press (1998) 280-285

[CS00] Cadoli, M. and Scarcello, F., Semantical and computational aspects of Horn approximations, *Artificial Intelligence* **119 (1-2)** (2000) 1-17

[Cur84] Curtis, C.W., *Linear algebra. An introductory approach*, Springer-Verlag (1984)

[DP92] Dechter, R. and Pearl, J., Structure identification in relational data, *Artificial Intelligence* **58** (1992) 237-270

[EG95] Eiter, T. and Gottlob, G., The complexity of logic-based abduction, *Journal of the ACM* **42 (1)** (1995) 3–42

[GJ83] Garey, M.R. and Johnson, D.S., *Computers and intractability : a guide to the theory of NP-completeness* , San Francisco : W.H. Freeman and Company (1983)

[JYP88] Johnson, D.S., Yannakakis, M. and Papadimitriou, C.H., On generating all maximal independent sets, *Inform. Process. Lett.* **27 (3)** (1988) 119–123

[KKS95] Kautz, H., Kearns, M. and Selman, B., Horn approximations of empirical data, *Artificial Intelligence* **74** (1995) 129-145

[KPS93] Kavvadias, D., Papadimitriou, C.H. and Sideri, M., On Horn envelopes and hypergraph transversals (extended abstract), in : *Proc. 4th International Symposium on Algorithms And Computation (ISAAC'93)* (Honk Kong), Berlin : Springer Lecture Notes in Computer Science **762** (1993) 399-405

[KS98] Kavvadias, D. and Sideri, M., The inverse satisfiability problem, *SIAM J. Comput.* **28 (1)** (1998) 152–163

[KSS00] Kavvadias, D.J., Sideri, M. and Stavropoulos, E.C., Generating all maximal models of a Boolean expression, *Inform. Process. Lett.* **74** (2000) 157–162

[Kha95] Khardon, R., Translating between Horn representations and their characteristic models, *Journal of Artificial Intelligence Research* **3** (1995) 349–372

[KR96] Khardon, R. and Roth, D., Reasoning with models, *Artificial Intelligence* **87** (1996) 187-213

[RK87] Reiter, R. and de Kleer, J., Foundations of assumption-based truth maintenance systems : preliminary report, in : *Proc. 6th National Conefrence on Artificial Intelligence (AAAI'87)* (Seattle, USA), Menlo Park : AAAI Press / MIT Press (1987) 183–188

[Sch78] Schaefer, T.J., The complexity of satisfiability problems, in : *Proc. 10th Annual ACM Symposium on Theory Of Computing (STOC'78)* (San Diego, USA), New York : ACM Press (1978) 216-226

[SK96] Selman, B. and Kautz, H., Knowledge compilation and theory approximation, *Journal of the ACM* **43 (2)** (1996) 193-224

[SH90] Stearns, R.E. and Hunt III, H.B., Power indices and easier hard problems, *Math. Systems Theory* **23** (1990) 209–225

[Val95] del Val, A., An analysis of approximate knowledge compilation, in : *Proc. 14th International Joint Conference on Artificial Intelligence (IJCAI'95)* (Montréal, Canada), San Francisco : Morgan Kaufmann (1995) 830-836

[Val00] del Val, A., The complexity of restricted consequence finding and abduction, in : *Proc. 17th National Conference on Artificial Intelligence (AAAI'00)* (Austin, Texas), AAAI Press / MIT Press (2000) 337–342

[ZH02a] Zanuttini, B. and Hébrard, J.-J., A unified framework for structure identification, *Information Processing Letters* **81** (2002) 335-339

[Z02a] Zanuttini, B., Approximating propositional knowledge with affine formulas, to appear in : *Proc. 15th European Conference on Artificial Intelligence (ECAI'02)* (Lyon, France), (2002)

[Z02b] Zanuttini, B., Des classes polynomiales pour l'abduction en logique propositionnelle (in French), *Proc. 8èmes Journées nationales sur la résolution de problèmes NP-Complets (JNPC'02)* (Nice, France), available from the author (2002)

# Abstracting Visual Percepts to Learn Concepts

Jean-Daniel Zucker*[1], Nicolas Bredeche[1,2], and Lorenza Saitta[3]

[1] LIP6 - Pole IA, Université Paris VI, 4 place Jussieu, 75252 Paris Cedex, France
{Jean-Daniel.Zucker;Nicolas.Bredeche}@lip6.fr

[2] LIMSI-CNRS, AMI, University Paris XI
Orsay, France
http://www.limsi.fr/

[3] Università del Piemonte Orientale, Dipartimento di Scienze e Tecnologie Avanzate
Corso Borsalino 54, 15100 Alessandria (Italy)
saitta@mfn.unipmn.it

**Abstract.** To efficiently identify properties from its environment is an essential ability of a mobile robot who needs to interact with humans. Successful approaches to provide robots with such ability are based on ad-hoc perceptual representation provided by AI designers. Instead, our goal is to endow autonomous mobile robots (in our experiments a Pioneer 2DX) with a perceptual system that can efficiently adapt itself to ease the learning task required to anchor symbols. Our approach is in the line of meta-learning algorithms that iteratively change representations so as to discover one that is well fitted for the task. The architecture we propose may be seen as a combination of the two widely used approach in feature selection: the Wrapper-model and the Filter-model. Experiments using the PLIC system to identify the presence of Humans and Fire Extinguishers show the interest of such an approach, which dynamically abstracts a well fitted image description depending on the concept to learn.

## 1 Introduction: Anchoring Symbols, Detecting, and Identifying Objects

Recent works in both Robotics and Artificial Intelligence have shown a growing interest in providing mobile robots with the ability to interact and communicate with humans. One of the main challenges in designing such robots is to give them the ability to perceive the world in a way that is useful or understandable to us. One approach is to give the robot the ability to identify physical entities and relate them to perceptual symbols that are used by humans (to refer to these same physical entities). To perform this task, the robot has to ground these symbols to its percepts (i.e., its sensor data). Recently, the term of ÒAnchoringÓ

* This work has been partially supported by a grant to the CNRS Jeune équipe "Découverte" (UMR 7606)

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 256–273, 2002.

[1] has emerged to describe the *building and maintenance of the connection between sensor data and the symbols used by a robot for abstract cognition.* As a matter of fact, anchoring is an important issue for any situated robot performing abstract reasoning based on physically grounded symbols. Anchoring plays an important role to communicate or relate to other robots [2] or humans [3].

There are tasks, such as object manipulation or functional imitation, where anchoring requires explicitly recognizing objects and localizing them in the three-dimensional space. Fortunately, such an *object recognition* task is not necessarily required to achieve anchoring. In applications such as human/object tracking, face and object identification, or grounded robot-human communication, *object identification* is enough. Informally, to recognize an object often requires from the robot both *identifying* from its percepts what an object is, and using a model of the object to localize it. This task has been studied for a few decades now and is known to be difficult in unknown environments [4]. On the contrary, identifying the sole presence of an object is simpler. Moreover, there exist many easy to use and reliable descriptions for characterizing the presence of an object. To identify the presence of a fire in a room, one does not have necessarily to recognize it. Smelling smoke, hearing cracks, feeling heat, seeing dancing shapes on a wall are different ways of identifying the presence of a fire. For an autonomous robot, the ability to identify objects is a first step towards more complex tasks. *Object detection* (detecting a fire) may be built by regularly checking whether the object is identified. Identifying objects is therefore a simple form of anchoring symbols (such as "fire") to its percepts.

In this paper, we are concerned with a practical task, where a PIONEER 2DX mobile robot has to rely on its limited vision sensors to anchor symbols such as ÒhumanbeingÓ, ÒmobilerobotÓ or ÒfireextinguisherÓ that it encounters while navigating in our laboratory. Anchoring is then used to support human/robot or robot/robot communication. For instance, an interaction may be engaged if a ÒhumanbeingÓ is identified, or a rescue operation may be initialized if a non-responding Ò PIONEER 2DXÓ is identified. Identifying a ÒfireextinguisherÓ may allow the robot to respond to a query formulated by a human. To design an autonomous robot, living in a changing environment such as our laboratory, with the identification ability described above is a difficult task to program. As such it is a good candidate for a Machine Learning approach, which may be easily recasted as a classical *concept learning task.* To teach the robot to anchor symbols using Machine Learning has proven successful [5]. To use machine learning techniques, the designer has to both define learning examples and a representation language based on the robot percepts to describe them.

It is clear that a great part of the success of the learning task per se depends on the representation chosen [6]. Having an AI designer providing the robots with an adequate representation has a major drawback: it is a fixed, ad-hoc representation. Any change of setting (a museum instead of an AI lab) may require a new perceptual description. In order to overcome this drawback, our main objective is to endow an autonomous robot with the ability to dynam-

ically abstract from its percepts different representations, well suited to learn different concepts. The intuitive idea is to have the robot explore a space of possible examples descriptions (with various colors, resolution, representation formalisms, etc.) so as to discover for each concept a well-fitted representation. The underlying intuition being that for anchoring the symbol "human being" a robot does not need the same visual resolution that might be necessary to anchor the symbol "power-plug".

Section 2 presents a concrete setting in which this problem occurs and pinpoints why adapting oneÕs representation may be useful to increase learning accuracy. In Section 3, related works about vision and anchoring in several research fields are quickly reviewed. Then, Section 4 explains our approach based on abstraction operators applied to visual information provided by the robot. Finally, a set of real world experiments describes the interest of such an approach and outlines the difference between two representations, each one fitted to a different concept (the presence of a human and the presence of a fire extinguisher).

## 2 Problem Settings

The practical task we are concerned with is part of a wider project called Microbes [7], whose goal is to have a colony of eight robots co-habit with AI researchers. We aim at providing each PIONEER 2DX autonomous mobile robot with the ability to identify -but not recognize- objects or living beings encountered in its environment. Each robot navigates during the day and when CPU resource is available it takes snapshots of its field of vision with its video camera. The snapshots are taken either randomly from time to time, or upon a specific human request[1]. At the end of each day, the robot may report to a supervisor and "ask" her/him what objects (whose symbols may or may not belong to a pre-defined lexicon) are to be identified on a subset of taken pictures[2]. It then performs a learning task in order to create or update the connection between sensory data and symbols which is referred to as the anchoring process. Figure 1 describes this process. The learning task associated to the anchoring is therefore characterized by a set of image descriptions and attached labels. It corresponds to a multi-class concept learning task. A key aspect of the problem lies in the definition of the learning examples (the images) used by the robot during the anchoring process. In effect, a first step in any anchoring process is to identify (relevant) information out of raw sensory data in order to reduce the complexity of the learning task. The PIONEER2DX mobile robot provides images thanks to its LCD video camera while navigating in the corridors. The images are $160 \times 120$ wide, with a 24 bits color information per pixel. Humans, robots, doors, extinguishers, ashtrays and other possible targets can be seen among the images as

[1] Thanks to *active learning* techniques, the robot may also take snapshots of scenes that appear to be interesting w.r.t. enhancing the detection accuracy of a known object (e.g. ambiguous images).

[2] Again *active learning* techniques may be used by the robot to select the most *informative* images.

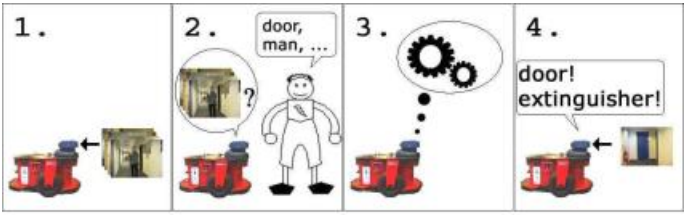



**Fig. 1.** The four steps toward lexicon anchoring. As a first step, the robot takes a snapshot of its environment, and a supervisor labels it with the interesting content. The robot tries to associate the provided label(s) with its percept, and, after a number of such steps take place, it shall be able to autonomously label a new environment.

**Fig. 2.** Two snapshots taken by the robot. Left: image labeled with "fire extinguisher" and "door". Right: image labeled with "human" and "door".

shown in Figure 2. All these possible targets, as they appear in the images, are of different shape, size, orientation and sometimes they are partially occluded. Finally, each image is labeled with the names of the occurring targets.

## 3 Changing the Representation of Images

### 3.1 Initial Perceptual Representation

We define the role of the robot's perceptual system as to extract *abstract percepts* out of *low-level percepts*, such as a set of pixels, from the video camera or sonar values. These abstract percepts provide a representation of the perceived world on which further computation will be based. They can be anything from sets of clustered colored regions to a matrix resulting from a Hough transform. The choice of a representation is motivated by finding a good trade-off that reduces the size of the search space and the expressiveness of the abstract percepts.

As mentioned in the previous section, the problem we consider is that of automatically finding a representation of a set of labeled images that is well adapted to the learning of concepts. Let us underline that our goal is not to achieve the best performance on the particular learning task mentioned in the previous section. To obtain the best performance would require that experts in the field build an ad-hoc representation for each concept to learn. On the contrary, we are interested in having a robot find by itself the good representation, so that,

**Table 1.** Datasets associated to the "human" and "fire extinguisher" concepts

| Concept to learn | # of positive examples | # of negative examples | Size of Dataset |
|---|---|---|---|
| Fire extinguisher | 175 | 175 | 20,1 Mb |
| Human being | 60 | 60 | 9,7 Mb |

if the context changes or the concept to learn is different, it has the ability to discover by himself the good level of representation. We therefore consider the representation provided by the sensors as an *initial* representation.

From the robot's point of view, each pixel from the camera is converted into a *low-level percept*. In the initial image representation, where each pixel is described by its position (x,y), its *hue* (the tint of a color as measured by the wavelength of light), its *saturation* (term used to characterize color purity or brilliance) and its *value* (the relative lightness and darkness of a color, which is also refereed to as "tone"). The initial description of an image is therefore a set of 19200 (160 x120 pixels) 5-uple (x,y,h,s,v). Each image is labeled by symbols following the process described in Section 2. The positive examples of a given concept (e.g., "presence of a fire extinguisher") to learn correspond to all images labeled positively for this concept. The negative examples are the images labeled negatively. The number of examples for two of the concepts we considered are given on Table 1. The initial representation of images, consisting of hundreds of thousands of pixels, is clearly a too low-level representation to be used by Machine Learning algorithms. We shall now analyze different representations that have been considered in the field of Vision or Image indexing from the Machine Learning point of view. These different representations will provide some directions for investigating automatic changes of representation to improve the learning accuracy.

### 3.2 Representation Languages in Machine Learning

In the traditional setting of Machine Learning, each object $x$ is represented by a *feature vector*, to which is associated a label $f(x)$. Let $\mathcal{X}$ be a feature vector space, and $\mathcal{Y} = \{\oplus, \ominus\}$. The induction task is to find a classifier $h : \mathcal{X} \to \mathcal{Y}$ which minimizes the probability that $f(x) \neq h(x)$ on a newly observed example $(x, f(x))$.

Within the *multiple instance* setting, objects are represented by *bags of feature vectors*. Feature vectors are also called *instances*, as in the traditional setting features may be numeric as well as symbolic features. The size of a bag $b$ is noted $\sigma(b)$ and may change from one object to another. Its instances are noted $b_1 \dots b_{\sigma(b)}$.

Within a *relational setting* (or Inductive Logic Programming (ILP) framework) the objects are represented by a set of components objects, their features, and relations between components. In ILP, Prolog facts are used to describe objects and Background Knowledge $B$ encodes deductive rules. To summarize, in Machine Learning the languages used to represent examples fall into three broad categories:

- Feature−Vector: the most widely used and for which efficient algorithms have been devised.
- Relational description: the most expressive representation but whose inherent complexity[8] prevents from efficient learning.
- Multiple−instance: an in-between representation, more expressive than feature−vector but for which efficient algorithms do exist.

### 3.3 Ad-Hoc Reformulation of the Initial Image Representation

Let us now characterize the various descriptions of the robot images from a Machine Learning point of view. The initial description may be seen as **a relational description** in the sense that an image is described by its components, the pixels, and their implicit relations given by the absolute position (x,y). As a matter of fact, region-growing algorithms, for example, do use the implicit information on neighbor pixels. The numerous drawbacks of such a low level representation (complexity to manipulate, size of the images) have lead researchers to come up with much simpler or more abstract representations of images.

A much more *abstract* representation is one where all pixels are replaced by a unique mega-pixel described by a hue, saturation and value that are averaged on the global image. This latter representation is also referred to as "global histogram representation" of the image and is widely used in image database retrieval systems. From a Machine Learning point of view it corresponds to a **feature/vector representation**. Each image is described by a vector of features such as the average hue, saturation and value, but standard deviations of these features might be added.

Although the global histogram representation performs particularly well in many situations (see [9] for a discussion), one of its drawback lies in its drastic loss of information. The position and color of pixels are lost. Another widely used approach in vision and image indexing is to abstract the image into a set of pixels (be it a region or not). This representation corresponds to a **multiple-instance representation** used in Machine Learning, where an image is considered as a bag of regions. Different types of sets are classically extracted:

1. counting pixels with the same color (a representation called "local histogram"): the information on the pixel positions is lost but not the color. This set is not a region per se, as pixels with the same color can be far apart.
2. grouping pixels based on their neighborhood into regions: new pixels at a smaller resolution are built. These regions can be recursively built.
3. grouping close pixels with similar color into regions: region growing techniques achieve such extraction, but are computationally costly and not always adapted to online processing.

These different representations are spread on the spectrum of possible abstractions for an image; they can be seen as obtained from the initial representation by an *abstract operator* that reduces the information contained in an image[10].

### 3.4 Dimensions of Abstraction

In the perspective of automatically exploring the set of possible representations of an image, we propose to identify particular operators and to experiment with them. There are countless operators that could be applied to an image hoping for more accurate learning. Operators changing the *contrast*, the *resolution*, the *definition*[3] are all possible candidates.

To improve the learning of concepts, we are interested in transformation that are abstractions in the sense that they decrease the quantity of information contained in the image[12]. The two main dimensions for abstraction that we shall study are:

- the resolution of the image, i.e., its granularity.
- the structure of the image, i.e., the smallest individually accessible portion of the image to consider, be it a pixel or a complex region.

For each of these dimensions, we have defined an abstraction operator: respectively, *associate* and *aggregate*. The associate operator consists in replacing a set of pixels with a unique (mega)pixel that has for its (h,s,v) values the average of the pixels that were associated. This operator is a built-in operator for the robot as it corresponds to a particular *sub-sampling*. The aggregate operator consists in grouping a set of pixels to form a region or a pattern. This operation is also referred to as "term construction" in the literature[13]. The region does not replace the pixels it is composed of, and therefore the resolution or granularity of the image is not changed. What changes is the structure of the image. The aggregate operator may be either data-driven (this is the case for region growing algorithms) or model based. For already mentioned reasons of efficiency required by the use of a robot we have considered an aggregate operator that is applied to contiguous pixels forming a particular shape. Figure 3 depicts the space of representation changes associated to these two operators.

We shall refer to the initial concept as *low-level percepts*[4]. The ones obtained after applying the abstract operators will be referred to as *abstract percepts*, since they will be used as percepts for further processing. For clarity, abstract percepts obtained by applying the aggregate operator will be referred to as *s-percepts* and ones obtained by applying the associate operator will be referred to as *r-percepts*[5] .

[3] The contrast measures the rate of change of brightness in an image; high contrast suggests content consisting of dark blacks and bright whites; medium contrast implies a good spread from black to white; and low contrast implies a small spread of values from black to white. The resolution is a measure of the proportion of the smallest individually accessible portion of a video image to the overall size of the image. The higher the resolution, the finer the detail that can be discerned. The definition corresponds to the clarity of detail in an image and is dependent upon resolution and contrast [11]

[4] We will depart from the traditional use of the term "ground" for the initial representation[14] as for a robot the notion of "grounding" corresponds to another notion

[5] s-percept, as in *structural percept* and r-percepts as in *resolution percept*

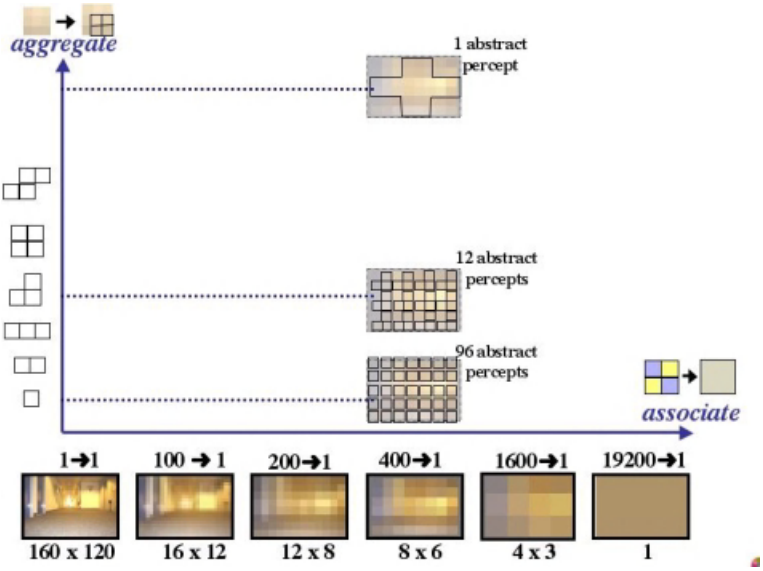



**Fig. 3.** The space of image representation obtained by applying the associate operator (changing the *resolution*) and the aggregate operator (changing the *structure*). Three examples of representations obtained after having applied the aggregate operators and the associate operator.

## 4 Automatically Changing the Representation for Learning

In the previous section two abstraction operators to change the representation of images were presented. The parameter of the associate operators we have considered is the number of pixels that are associated to form a (mega)pixel. The parameter of the aggregate operator is the pattern or region structure.

With respect to the learning task described in Section 2, a key issue is to analyze the impact of representation changes on learning. The main question is related to the choice of one operator and its parameters. In Machine Learning, the abundant literature on feature selection shows that approaches fall in two broad categories: the `wrapper` and the `filter` approach. Intuitively, the `wrapper` approach uses the performance of the learning algorithm as a heuristic to guide the abstraction. The `filter` approach uses a a priori knowledge to select appropriate abstractions. In Machine Learning the combination of this two approaches has not yet been explored. In the following, we present how these two approaches can be combined. As it is an approach that attempt to learn from the learning process itself it is also referred to as a *meta-learning* approach.

Since resolution changes the information contained in an image, we have used an information-based filter approach to choose an a-priori good resolution to start with for learning. The filter approach is therefore used to explore the

horizontal dimension of the space of Figure 3. To explore different possible patterns to apply with the aggregate operator, we have used a `wrapper` approach. The PLIC system is the result of the combination of these two approaches.

### 4.1 A Filter-Based Exploration of the Abstract Spaces

As mentioned previously, there are two dimensions along which to apply abstraction operators: image resolution (association) and structural composition (aggregation). While the structural composition dimension is explored by a wrapper approach, the resolution can be approached by a filter approach.

Let us suppose that a snapshot with $N$ pixel is taken, and that we are looking for the localization of an object with $R$ pixels. Let us suppose, for the sake of simplicity, that each pixel is either activated (has an intensity value 1) or not activated (has an intensity value 0). The level with $N$ pixels is the more detailed one, whereas the maximum confusion is reached when the activity value is averaged over all the $N$ pixel, obtaining thus a single, large percept. Let $\tau$ be the homogeneous value of this largest percept. If we assume that there is no other object in the snapshot, we will have : $\tau = \mathrm{R/N}$

Moreover, let $\sigma$ be the standard deviation of the intensities over the $N$ pixels. If groups of $K$ pixels are associated into a single percept, its average intensity will be computed from the intensities of the component pixels, and the object we are looking for will appear more or less blurred, depending on how many activated pixels are included in the $K$ ones. Let us assume that the object can still be discovered if its average intensity is greater than $\alpha\tau$, where $\alpha$ is a number greater than one, which depends on the sensor sensibility. For instance, if we can distinguish objects whose average intensity differs by the global average $\tau$ by $p$ standard deviations, we will have: $\alpha = 1 + \mathrm{p} \times \sigma \ / \ \tau$

Let us now consider a generic percept containing an association of K pixels. Depending on the position of this percept w.r.t. the object, its average intensity will change. Among all the positions, there will be at least one that reaches a maximum of intensity, and will allow the object to be optimally identified. If there is only an object in the picture, the question is whether the object is present or not. Notice that the average intensity $\tau$ can be generated, if the object is not present, by R activated pixels more or less randomly scattered in the snapshot. If $K \leq R$, the optimal new percept is included in the object picture, and its average intensity will be 1. If $K > R$, the optimal percept will include R pixels with intensity 1 and (K - R) pixels with intensity 0. Then, the optimal percept intensity will be: $\mathrm{I(K)} = \mathrm{R/K}$. In Figure 4 an example of the graph of the optimal intensity I(K) vs. K is reported for N = 1000 and R = 200. In order to locate the object, the percept intensity must be greater than $\tau$: $I(K) \geq \alpha\tau$

From the above condition we obtain a maximum number of pixels for the new percept:

$$K \leq N[\tau/(\tau + p\sigma)] \tag{1}$$

It is clear the intensity of the new percept tends to $\tau$ when $K$ tends to $N$. Another interesting aspect to be considered is the number of optimal percept

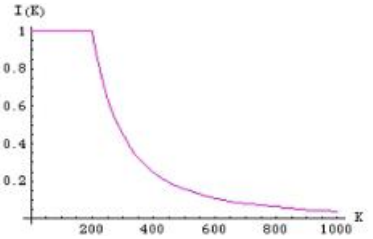



**Fig. 4.** Optimal percept intensity I(K) to detect a generic percept containing an association of K pixels (for N = 1000 and R = 200).

that can be present in the image when K changes. Assuming that $\sqrt{X}$ is the order of the linear dimension of an object whose area spans X adjacent pixels, the probability P(K) of randomly extracting an optimal percept of size K can be roughly estimated as follows :

$$P(K) \sim [(\sqrt{R} - \sqrt{K} + 1)/(\sqrt{N} - \sqrt{K} + 1)]^2 \text{ if K} \leq \text{R}$$
$$P(K) \sim [(\sqrt{K} - \sqrt{R} + 1)/(\sqrt{N} - \sqrt{R} + 1)]^2 \text{ if K} > \text{R}$$

In Figure 5(left) an example of the P(K) graph vs. K is reported for N = 1000 and R = 200. It is clear that reducing K the optimal intensity increases, but, at the same time, the number of percepts with that intensity decreases, making more difficult (more percepts are to be examined) to locate the object, if it is present in the picture. In Figure 5(right) the product I(K) × P(K) is reported for the

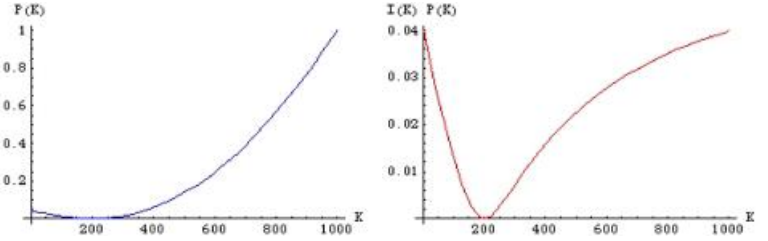



**Fig. 5.** (left) The probability P(K) of randomly extracting an optimal percept of size K. (right) The product I(K) × P(K).

sake of illustration. The above reasoning can be extended to values associated to the pixels different from a binary intensity. For example, in the application considered in this paper, one of the averaged value can be the color hue. In this case, we have verified that the red color of the fire extinguisher, for instance, can be detected by the system at a resolution (8 × 6), which does not allow the color to be detected by the human eye.

### 4.2 A Wrapper-Based Exploration of the Abstract Spaces

Once a resolution has been selected by the filter-based approach described in the previous sub-section, the wrapper-based component of the PLIC system explores different image structure iteratively. An initial structure is chosen, and the image is reformulated in a multiple-instance representation using this structure; then, the concepts are learnt using this representation. Based on the results on cross-validation[6] of the learning algorithm, a new structure is devised. As for now, the search for a good structure is done in an exhaustive manner from the simplest one (i.e., one pixel at the chosen resolution) and exploring all the connected shapes of $k$ pixels before increasing $k$. The following figure is a synthesis of this wrapper approach. The multiple instances rule learner RIPPERMI[15] was used

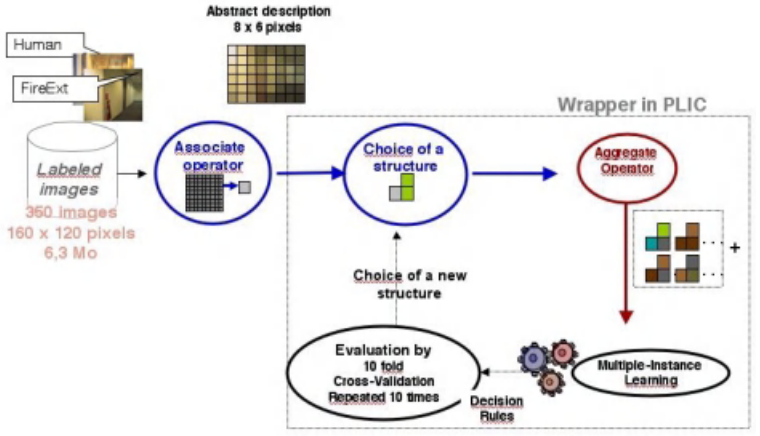



**Fig. 6.** The PLIC system Wrapper component.

on the descriptions obtained from these images with a ten-fold cross validation. Moreover, each experiment is repeated 10 times in order to get a good approximation of the results. RIPPERMI returns a set of rules that covers the positive examples

## 5 Experiments

### 5.1 Experimental Setup

To evaluate the interest of abstracting visual percepts from a Machine Learning point of view, a number of different experiments have been carried out. The experiments presented are based on the images acquired by a PIONEER2DX

[6] a widely used data-oriented evaluation of the learning generalization error that consists in dividing the learning set into a learning set and a training set

**Table 2.** PLIC results on learning the "human" concept

| Resolution | Accuracy in % | std.dev | CPU time (s) |
|---|---|---|---|
| 1x1 | 64,02 | 1,29 | 0,03 |
| 4x3 | 59,11 | 1,09 | 0,13 |
| 8x6 | 62,47 | 1,27 | 0,79 |
| 16x12 | 67,54 | 1,23 | 4,03 |
| 32x24 | 66,45 | 1,45 | 19,12 |
| 64x48 | 65,48 | 1,28 | 87,33 |

**Table 3.** PLIC results on learning the "fire extinguisher" concept

| Resolution | Accuracy % | std.dev | CPU time in secs |
|---|---|---|---|
| 1x1 | 62,51 | 0,85 | 0.39 |
| 4x3 | 59,58 | 0,82 | 1.62 |
| 8x6 | 64,18 | 0,89 | 9.22 |
| 16x12 | 70,98 | 0,86 | 37.68 |
| 32x24 | 75,02 | 0,96 | 128.31 |

mobile robot. The targets (be it a human or a fire extinguisher) as they appear in the images are different in shape, size, orientation and are sometimes partially occluded. Labeling with the names of the occurring targets was done by a supervisor (as explained in Section 2), and a noisy set of labels was produced as well (wrong labels were given on purpose). Two sets of experiments are presented, the first one illustrates the impact of the operator aggregate used by the wrapper-based component of PLIC and the second the impact of the associate operator used by its filter-based component. For each of these experiments the results for the concept "human" and "fire extinguisher" are given.

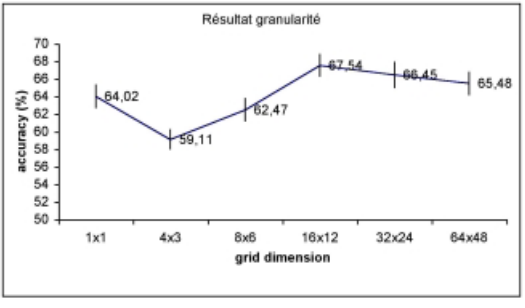



**Fig. 7.** Experiments with the "human" concept

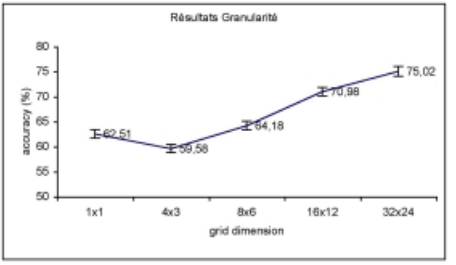



**Fig. 8.** Experiments with the "fire extinguisher" concept

### 5.2 Evaluating Automatic Changes of Granularity

The results obtained by the system PLIC are presented in Tables 3 and 2 below. Figures 8 and 7 respectively plot the evolution of the learning accuracy depending on the resolution chosen for the extinguisher class and the human class.

The evolution of the learning accuracy for the extinguisher class vs. the resolution shows that more complex resolutions are much more appropriate that the simpler ones (i.e., the histogram representation). As a matter of fact, the accuracy is enhanced by 12.51%, from 62.51% to 75.02%. However, this improvement is not linear over the set of possible resolutions, since the accuracy is better for the original $1 \times 1$ resolution than for the $4 \times 3$ resolution. This can be explained by the fact that the histogram representation is somewhat fitted to capture some relevant information for image description [9]. As a consequence, using a more complex representation may cause the accuracy to decrease since the enhanced search space makes it more complicated to learn while it may not help to better discriminate the target concept.

Resolution enhancement proves to be profitable as soon as a $8 \times 6$ resolution and the accuracy improves in a nearly linear way from this point. Now, if we bring our attention to the evolution of the learning accuracy for the human class, we can see that the learning curve is about the same, with a smaller amplitude, except that at some point the accuracy reaches a maximum (67.54% with the $16 \times 12$ resolution), and then starts to decrease. This should tend to prove that the $16 \times 12$ resolution is the most fitted resolution for this concept, in this environment, given what examples have been observed. More complex resolutions cannot achieve a better representation for this concept, or at least the tradeoff between expressiveness and complexity is not relevant anymore. As a matter of fact, we can extrapolate to say that if the resolution is enhanced, accuracy should tend to reach 50% at some point, that is random prediction.

However the general shape of the accuracy curves for both concepts seem to share the same properties about the trade off between expressiveness and complexity, experiments shows that the best fitted granularity depends on the concept to learn.

### 5.3 Experiments on Automatic Changes of Structure

PLIC's wrapper tool was used to generate nine different s-percept's structural configurations that are shown on figure 9. Each structural configuration is then applied from every single r-percept to generate a learning sets based on a $8 \times 6$ image resolution of each of the images of the image database, which is not the best resolution for either concepts but provides a common basis for evaluating learning accuracies. Tables 5 and 4 respectively show the results for the human class and the fire extinguisher class.

Results from the experiments show that in both case, the highest accuracy is achieved by one of the most complex structural configurations (64.81% for the extinguisher class, 71.56% for the human class). The benefits of structural reformulation is more relevant for the human class with a 3.5% gain than for the extinguisher class, where it is nearly useless. This can be explained by the intrinsic properties of both concepts: detecting a human may require expressing relations between parts (e.g. *head on body*) while an extinguisher is often viewed as a uniform rectangle red shape in the environment.

In these experiments, this structural configuration is quite simple because of the low number of abstract percepts considered. However, human occurrence detection was achieved by the robot more than 7 times out of 10 thanks to less than 48 s-percepts.

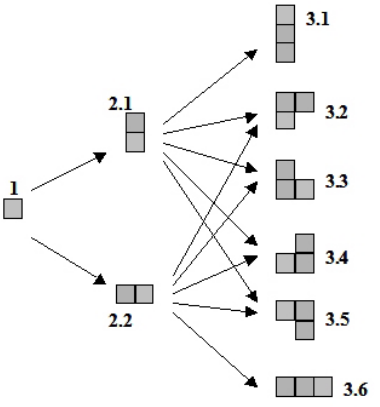



**Fig. 9.** The nine structural configurations generated by the Wrapper

## 6 Related Works

In robotics, reliable information is most of the time provided by simple stimuli, since vision processing is difficult to handle and costs much of the processing time unless dedicated chips are used. Works on anchoring a lexicon are usually focused

**Table 4.** PLIC results on learning the "fire extinguisher" concept and changing structure

| StructureName | Accuracy % | std.dev | CPU time (s) |
|---|---|---|---|
| 1 | 64,18 | 0,89 | 10,43 |
| 2.1 | 62,04 | 0,715 | 7,475 |
| 2.2 | 63,795 | 0,83 | 10,185 |
| 3.1 | 64,335 | 0,695 | 9,02 |
| 3.2 | 64,405 | 0,755 | 8,655 |
| 3.3 | 63,21 | 0,715 | 9,465 |
| 3.4 | 63,46 | 0,75 | 9,475 |
| 3.5 | 64,81 | 0,77 | 9,615 |
| 3.6 | 64,175 | 0,79 | 12,08 |

**Table 5.** PLIC results on learning the "human" concept and changing structure

| Structure name | Accuracy % | std.dev | CPU time (s) |
|---|---|---|---|
| 1 | 68,06 | 1,35 | 1,44 |
| 2.1 | 70,645 | 1,375 | 1,785 |
| 2.2 | 65,665 | 1,355 | 2,27 |
| 3.1 | 66,575 | 1,24 | 2,095 |
| 3.2 | 66,48 | 1,38 | 2,36 |
| 3.3 | 67.01 | 1,335 | 2,37 |
| 3.4 | 68.945 | 1,24 | 2,235 |
| 3.5 | 71,56 | 1,17 | 2,235 |
| 3.6 | 64,87 | 1,36 | 2,58 |

on higher level cognition and adaptation, such as language games [16], and thus do not deal with complex scenes. However, when it comes to applications with complex vision processing needs, works in robotics share much with works from other communities, such as object detection and tracking or image indexing, where there is an impressive body of literature.

On the one side, tracking applications are mostly embedded in systems using a video camera, with or without stereo vision. A known approach to tracking is multiple hypothesis tracking, where temporal information is used to choose between hypothesis. A more recent approach relies on a detection method where tracking is the process that checks the spatial coherence through time of a single hypothesis given by a detection algorithm [17]. Both approaches usually rely on model-based detection, and use image transforms to detect shapes along with correlation to cope with the occlusion problem. Some tracking applications in robotics also implement such approaches, and make use of the robot capability to follow the target.

On the other side, image indexing is concerned with classifying an image based on its content, without precisely identifying the location of the target concept in the image. Approaches to image indexing include model-based classifica-

tion, images description using Fourier transforms, wavelets[18], etc. A popular approach is based on matching sets of connected color regions between images [19]. The goal is to find instances of a given spatial configuration between regions extracted from the images (i.e., using a region growing algorithm). The main drawbacks of this approach are the imprecision of region growing techniques, and the cost of the matching phase between undirected planar graphs, representing sets of connected regions. However, as mentioned previously good classification results also also achieved by simply comparing the global color histogram of each image [9]. In mobile robots, image classification into categories is used for creating landmarks or for navigating. In this case, image indexing using global color histograms is particularly well fitted because it classifies quickly the whole image.

In the above mentioned domains, few works are concerned with adaptive issues, which is a main concern in robotics since [20]. Extending Marr's visual object recognition paradigm [21], where recognition is a bottom-up process, perceptual adaptation is considered as an important component for any perception-based cognitive activities, and initiated many works concerning perceptual learning [22], active perception, perceptual attention, etc.. However, even though these concerns are of primary importance in the field of psychology of perception, and could provide an interesting framework for artificial perception, we do not know of any anchoring application for a mobile robot.

## 7 Conclusion

In this paper we have addressed the problem of using automatic abstraction of visual percepts by an autonomous mobile robot to improve its ability to learn *anchors*[1]. This work finds its application in a real-world environment within the MICROBES multi-robots project [7], where anchors provides a basis for communication between the Pioneer 2DX robot and its human interlocutor. In the approach we proposed the robot starts with the initial low-level representation of the images it perceives with its LCD video camera, and iteratively changes their representation so as to improve the learning accuracy. Between the low-level pixel representation and a global histogram representation there is an immense space of possible representations. To explore part of this abstract space of representation we have identified two operators. A first one changes the resolution and loose information by averaging the color of squares of pixels. A second one that groups pixels without changing the resolution.

To guide the exploration of the space of possible abstractions, we have combined in the PLIC system two approaches, one based on a priori considerations, and one using the learning results. From a Machine Learning point of view, this architecture corresponds to the combination of the two widely used approaches in feature selection: the Wrapper-model [23] and the Filter-model [24]. The set of experiments that have been conducted show that both operators do impact on the learning accuracy. It is clear that the representation found is not optimal as it is very possible that both abstraction operators might interact. Further

work includes taking into account the interaction between the aggregate and associate operator. Nevertheless, it is interesting to notice that the best resolution and structure (sort of coordinates in the abstract space) found by the system depends of the concept. Since less low-level information is required to detect the presence of a human than a fire extinguisher, it is not surprising that the optimum resolution is different. It is also clear that as the number of examples increase, different reformulation might perform better. Creating high-level abstract percepts does not only improve accuracy, it makes object detection faster for the robot. This is true as long as the abstraction process does not itself takes too much time. This is a known trade-off in the field of abstraction [25]. As a matter of fact, abstracting regions by using region growing algorithm [26] was a candidate abstract operators but its computation is too costly for online detection.

This study shows that for learning anchors, an approach that periodically searches for the most accurate representation, given the examples at hand, is a promising direction. Moreover, it appears that for each anchor that needs to be learnt, different abstractions might be more appropriate. These findings, although preliminary, raises several questions with respect to the robot architecture. A central question for any lifelong learning system, integrating abstraction abilities, is to decide whether to continue to *exploit* its current representation or *explore* new representations at the risk of loosing resources if no better ones is found.

**Acknowledgments.** We would like to thank the anonymous reviewers for their helpful comments and remarks (especially the ones concerning the possible interactions between abstraction operators) as well as Alexis Drogoul in charge of the MICROBES Project.

## References

1. Coradeschi, S., Saffiotti, A.: Anchoring symbols to sensor data: preliminary report. In: Proceedings of AAAI-2000, Austin, Texas (July 2000)
2. Steels, L.: The talking heads experiment. volume 1. words and meanings. In: Antwerpen. (1999)
3. Thrun, S., Bennewitz, M., Burgard, W., Cremers, A., Dellaert, F., Fox, D., H hnel, D., Rosenberg, C., Roy, N., Schulte, J., Schulz, D.: Minerva: A second generation mobile tour-guide robot. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA). (1999)
4. Stone, J.: Computer vision: What is the object? In: Prospects for AI, Proc. Artificial Intelligence and Simulation of Behaviour. Birmingham, England., IOS Press, Amsterdam. pages 199–208 (1993)
5. Klingspor, V., Morik, K., Rieger, A.D.: Learning concepts from sensor data of a mobile robot. Machine Learning **23** (1996) 305–332
6. Saitta, L., Zucker, J.D.: A model of abstraction in visual perception. In: Applied Artificial Intelligence. 15(8): 761-776. (2001)
7. Picault, S., Drogoul, A.: The microbes project, an experimental approach towards open collective robotics. In: Proc. of the 5th International Symposium on Distributed Autonomous Robotic Systems, Springer-Verlag Tokyo Inc. (2000)

8. Giordana, A., Saitta, L.: Phase transitions in relational learning. Machine Learning Journal **41** (2000) 217–
9. Stricker, M., Swain, M.: The capacity and the sensitivity of color histogram indexing. In: Technical Report 94-05, Communications Technology Lab, ETH-Zentrum. (1994)
10. Saitta, L., Zucker, J.D.: Semantic abstraction for concept representation and learning. In AAAI), S.i.P.b., ed.: Symposium on Abstraction, Reformulation and Approximation (SARA98), Asilomar Conference Center, Pacific Grove, California (1998)
11. Drury, S.: A Guide to Remote Sensing. The Kluwer International Series on Information Retrieval, 11, Oxford (1990)
12. Saitta, L., Zucker, J.D.: A model of abstraction in visual perception. Applied Artificial Intelligence **15** (2001) 761–776
13. Giordana, A., Saitta, L.: Abstraction: a general framework for learning. In: Working notes of the AAAI Workshop on Automated Generation of Approximations and Abstraction, Boston, MA (1990) 245–256
14. Sacerdoti, E.: Planning in a hierarchy of abstraction spaces. Artificial Intelligence **5** (1974) 115–135
15. Chevaleyre, Y., Zucker, J.D.: A framework for learning rules from multiple instance data. In: Proc. European Conference on Machine Learning (ECML2001). (ECML2001)
16. Steels, L.: The origin of syntax in visually grounded robotic agents. In: Proceedings of IJCAI97, Morgan Kaufman Pub. Los Angeles. (1997)
17. Beymer, D., Konolige, K.: Real-time tracking of multiple people using continous detection. In: Proceedings of the International Conference on Computer Vision (ICCV'99). (1999)
18. Wang, J.Z.: Integrated Region-based Image Retrieval. The Kluwer International Series on Information Retrieval, 11, Oxford (2001)
19. Hsieh, I., Fan, K.: Color image retrieval using shape and spatial properties. In: ICPR00, Vol.I: pp 1023-1026. (2000)
20. Brooks, R.: Intelligence without representation. Artificial Intelligence **47** (1991) 139–159
21. Marr, D.: Vision. Freeman and Co., Oxford (1982)
22. Goldstone, R.L.: Perceptual learning. In: Annual Reviews of Psychology. 49:585-612. (1998)
23. Kohavi, R., John, G.: The wrapper approach. In: Feature Selection for Knowledge Discovery and Data Mining, H. Liu and H. Motoda (eds.), Kluwer Academic Publishers, pp33-50. (1998)
24. Kohavi, R., Sommerfield, D.: Feature subset selection using the wrapper method: Overfitting and dynamic search space In: International Conference on Knowledge Discovery and Data Mining. (1995)
25. Giunchiglia, F.: Using abstrips abstractions where do we stand ? Artificial Intelligence Review **13** (1996) 201–213
26. Rehrmann, V., Priese, L.: Fast and robust segmentation of natural color scenes. In: Asian Conference on Computer Vision, Hongkong, China. (1998)

# PAC Meditation on Boolean Formulas

Bruno Apolloni[1]*, Fabio Baraghini[1], and Giorgio Palmas[2]

[1] Dip. di Scienze dell'Informazione, Università degli Studi di Milano
[2] ST Microelectronics s.r.l. Agrate Brianza (Mi) - Italy

**Abstract.** We present a Probably Approximate Correct (PAC) learning paradigm for boolean formulas, which we call PAC meditation, where the class of formulas to be learnt are not known in advance. On the contrary we split the building of the hypothesis in various levels of increasing description complexity according to additional constraints received at run time. In particular, starting from atomic forms constituted by clauses and monomials learned from the examples at the 0-level, we provide a procedure for computing hypotheses in the various layers of a polynomial hierarchy including k_term-DNF formulas at the second level. Assessment of the sample complexity is based on the notion of sentry functions, introduced in a previous paper, which extends naturally to the various levels of the learning procedure. We make a distinction between meditations which waste some sample information and those which exploit all information at each description level, and propose a procedure that is free from information waste. The procedure takes only a polynomial time if we restrict us to learn an inner and outer boundary to the target formula in the polynomial hierarchy, while an access to an NP-oracle is needed if we want to fix the hypothesis in a proper representation.

## 1 Introduction

PAC learning is a very efficient approach for selecting a function within a class of Boolean functions (call them concepts) on the basis of a set of examples of how this function computes [1]. In this paper we will consider an extension of this approach to the case that the class of concepts is not known at the beginning. Rather we receive requisites of the class a little at a time in subsequent steps of the learning process. Thus we must have at runtime a twofold care of:

1. correctly updating current knowledge on the basis of new requisites, so that the approximation of the hypotheses on the final concept is not compromised, and

2. suitably reinterpreting examples in the light of the current knowledge, so that only their essential features are focused on, without neither missing necessary data nor recording unuseful details.

We hit these targets in learning boolean formulas through a multi-level procedure that we call *PAC-meditation*:

- at the first level we have two sets of positive and negative examples. From subsets of equally labelled examples we compute partial consistent hypotheses. Namely each hypothesis is consistent with a part of the positive examples and all negative

* Corresponding author: e-mail apolloni@dsi.unimi.it

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 274–281, 2002.

examples, or vice-versa. The criterion is that the union of the hypotheses coming from positive subsets and the intersection of the other ones form two nested regions delimiting the gap where the contours of suitable consistent hypotheses are found. In Fig. 1 the gap is represented by the dashed area. We distinguish a gray region embedded in a white dashed one. Let us focus for a moment on the widest area (contoured by thin curves), which we call 0-level gap. It is delimited on the inside by a (non dashed) region we call *inner border* and, analogously, by the *outer border*.

– at further abstraction levels the partial consistent hypotheses of the immediately preceding level play the role of labelled examples, where the sampled data are substituted by formulas and the positive and negative labels are substituted by a flag which denotes whether these formulas belong to the inner or outer borders. A new pair of borders are constructed running the same procedure on the so represented examples (and are contoured by bold lines in Fig. 1). Not far from what happens in the human mind, an actual benefit comes from these level jumps in case of suitable definition of the classes of formulas in the new borders. These classes induce new links between the formulas, with the twofold effect of reducing both the degrees of freedom of the final class of hypotheses, thus lowering the sample complexity of the learning problem, and narrowing the interstice between the borders, thus simplifying the search for a final hypothesis.

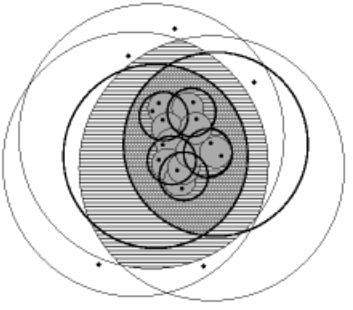


**Fig. 1.** Inner and outer borders, in the sample space, of a concept at two abstraction levels. Inner borders are delimited by the union of formulas bounded by positive examples (gray circles with thin contour at ground level), outer borders by the intersection of formulas bounded by negative examples (white circles with thin contour at ground level). Bold lines describe higher level formulas. Bullets: positive examples; rhombuses: negative examples.

The consistency constraint binds the whole learning process, which coincides in this respect with an efficient watching on the part of training examples sentineling that borders do not trespass forbidden points. This functionality represents the points' information content which will be managed optimally, i.e. without *information waste*. Passing from one symbolic level to another, properties about these points become points in a new functional space (call them hyperpoints within a higher abstraction level), that are useful,

in own turn, for building new properties, i.e. metaproperties on the original example space.

In this way our procedure puts a bridge between inductive and deductive learning. The atomic formulas at the first level are inductively learnt from examples [1], then they are managed through special deductive tools. This is a true different acception of agnostic learning. With the general understanding that it is very difficult to know a priori the class of the goal concept or even the set of involved variables [2], many authors infer its functional shape directly from the data within paradigms like boosting [3] or other kind of modular learning [4]. Their approaches share with the most elmentary ones like decision trees [5] or Rulex [6] the idea that this shape comes uniquely from a best fitting of the training set data. Our approach aims at building a concept by improving elementary formulas using in a lolgical way pieces of symbolic knowledge coming from an already achieved experience. This allows for a more complex and well funded management of the trade-off between class complexity and error rate clearly sinthetised by Vapnik in the problem of the structural risk minimization [7]

For lack of space, the exposition proceeds through a series of definitions and theorems whose proof is deferred elsewhere. In parfticular, in Sect. 2 we review the PAC learning theory within a new statistical framework, while Sect. 3 is devoted to introduce the conceptual framework of PACmeditation and the related theoretical results. A very short numerical section concludes the paper.

## 2 PAC Learning Theory Revisited

A very simple way we found for discussing of the statistical properties of a learning procedure is the following [8]. We have a labeled sample $\mathbf{Z}_m = \{(X_i, b_i),\ i = 1, \ldots, m\}$ where X takes values in $\mathfrak{X}$ [1] and $b_i$ are boolean variables. We assume that for every $M$ and every $\mathbf{Z}_M$ an $f$ exists in a boolean class $C$, call it *concept* $c$, such that $\mathbf{Z}_M = \{(X_i, c(X_i)), i = 1, \ldots, M\}$, and we are interested in the measure of the symmetric difference $U_{c \div h}$ between another function computed from $\mathbf{Z}_m$, that we denote as *hypothesis* $h$, and any such $c$ (i.e. tyhe set of points where we will answer 0 using $h(x)$ while the correct answer is $c(x) = 1$ or vice versa, see Fig. 2).

Actually, for fixed sample $\mathbf{Z}_m$ we can have different populations $\mathbf{Z}_M$, hence different $c$'s explaining them. These are related however to the explanation $h$ we found for the sample, and we work precisely on this relation for computing the distribution law of the random variable $U_{c \div h}$ as a function of the randomm suffix $\mathbf{Z}_M$ of a given sample $\mathbf{z}_m$. This relation is very similar to the one between sample and population properties of a Bernoulli variable, as in both cases we work with 0/1 assignments. But here we need some sampled points – which we call *(outer) sentry points* [9] – to recognize that the probability measure of the error domain is less than a given $\varepsilon$. These points are assinged by a *sentinelling function* $\mathbf{S}$ whose formal definition is given in [9], to each concept of a class in such a way that : i. they are external to the concept $c$ to be sentinelled and internal to at least one other including it, ii, each concept $c'$ including $c$ has at least one

[1] By default capital letters (such as $U$, $X$) will denote random variables and small letters ($u$,$x$) their corresponding realizations; the sets the realizations belong to will be denoted by capital gothic letters ($\mathfrak{U}$, $\mathfrak{X}$).

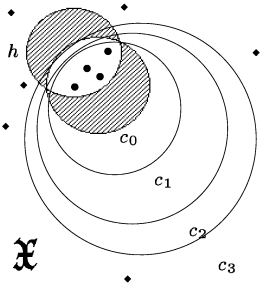



**Fig. 2.** A PAC learning framework. $\mathfrak{X}$: the set of points belonging to the cartesian plane; $c$: a concept from the concept class of circles; $h$: a hypothesis from the same concept class; bullets: 1-labeled (positive) sampled points; rhombuses: 0-labeled (negative) sampled points. Line filled region: symmetric difference.

of the sentry points of $c$ either in the gap between $c$ an $c'$ or outside of $c'$ and distinct from the sentry points of $c'$, and iii. they constituted a minimal set with these properties. An upper bound to the cardinality of these points is represented by the detail $D_C$ of a concept class. For instance, the class $C$ on $\mathfrak{X} = \{x_1, x_2, x_3\}$ whose concepts are

$$\begin{array}{rccc} & x_1 & x_2 & x_3 \\ c_1 = & \ominus & \ominus & - \\ c_2 = & \ominus & + & + \\ c_3 = & + & \ominus & + \\ c_4 = & + & + & + \end{array} \qquad \begin{array}{rccc} & x_1 & x_2 & x_3 \\ c_1 = & - & - & \ominus \\ c_2 = & \ominus & + & + \\ c_3 = & + & \ominus & + \\ c_4 = & + & + & + \end{array}$$

where "+" denotes an element $x_j$ belonging to $c_i$, "−" an element outside $c_i$ and $\ominus$ a sentry point, has $\mathrm{D}_C = 2$. A worst case $\mathbf{S}$ is: $\mathbf{S}(c_1) = \{x_1, x_2\}$, $\mathbf{S}(c_2) = \{x_1\}$, $\mathbf{S}(c_3) = \{x_2\}$, $\mathbf{S}(c_4) = \emptyset$. However a cheaper one is $\mathbf{S}(c_1) = \{x_3\}$, $\mathbf{S}(c_2) = \{x_1\}$, $\mathbf{S}(c_3) = \{x_2\}$, $\mathbf{S}(c_4) = \emptyset$. Further examples can be found in [9]. In particular here we will refer to classes of concepts $C \div C$ made up of the symmetric differences $c_i \div c_j$ between concepts belonging to a same class $C$ and its detail $D_{C,C}$.

A learning algorithm is a procedure $\mathcal{A}$ to generate a family of hypotheses $h_m$ with their respective $U_{c \div h_m}$ converging to 0 in probability with the sample size $m$.

**Lemma 1.** *For a space $\mathfrak{X}$ and an unknown probability measure $\mathrm{P}$ on it, assume we are given i) a concept class $C$ on $\mathfrak{X}$ with $D_{C,C} = \mu$, ii) a sample $\mathbf{Z}_m$ drawn from the fixed space and labeled according to a $c \in C$ labeling an infinite suffix $\mathbf{Z}_M$ of it, and iii) a fairly strongly surjective function $\mathcal{A} : \{\mathbf{Z}_m\} \to C$ misclassifying at most $t \in \mathbb{N}$ points of total probability not greater than $\rho$.*

*In case $m \geq \max\left\{\frac{2}{\varepsilon}\log\frac{1}{\delta}, \frac{5.5(\mu+t-1)}{\varepsilon}\right\}$ $\mathcal{A}$ is a learning algorithm for $C$ such that $\mathrm{P}^{(M)}\left(U_{c \div \mathcal{A}(\mathbf{Z}_m)} \leq \max\{\rho, \varepsilon\}\right) \geq 1 - \delta$.*

*In case $m < \frac{1-\varepsilon}{\varepsilon}\log\frac{1}{\delta}$ no learning algorithm exists satisfying the above probabilistic inequality on the measure of the symmetric difference.* □

The main lesson we draw from the above discussion is that, when we want to infer a function we must divide the available examples in two categories, the relevant ones

and the mass. Like in a professor's lecture, some, the former, straight fix the ideas, thus binding the difference between concept and hypothesis. The latter are redundant; but if we produce a lot of examples we are confident that a sufficient number of those belonging to the first category will have been exhibited.

## 3 PAC-Meditation

If we do not know C in advance we propose a procedure to discover it progressively. Its block diagram is shown in Fig. 3. Given a set of positive and negative examples the procedure core consists in the iterated implementation of an abstraction module made up of two steps: i. a *Symbols' jump*, where we introduce new symbols to describe (Boolean) properties on the points; and ii. a *Reduction* step for refining these properties. Namely, we start considering a set of minimal hypotheses about the goal formula that are consistent with positive examples and maximal for the negatives ones. Thus a second step is devoted to broadening or narrowing these hypotheses with: i. the constraint of not violating the examples consistency and ii. the scope of narrowing the gap between the union of minimal hypotheses (the mentioned inner border) and the intersection of the maximal hypotheses (the mentioned outer border). This happens at zero level. To increase the abstraction level we may restart the two steps after assuming the minimal hypotheses as positive (hyper)points at 1-level, maximal hypotheses as negative hyperpoints, and searching for new hypersymbols to describe properties on these new points. To avoid tautologies, the new abstraction level must be enriched by pieces of symbolic knowledge that are now available about the properties we want to discover, and translate in additional constraints in rebuilding the borders. Once we are satisfied with the abstraction level reached (or simply do not plan on achieving new formal knowledge), the level test in Fig. 3 addresses us to the Synthesis step. Here we collapse the two borders into a single definite formula lying between them which we assume as representative of the properties on the random population we observed.

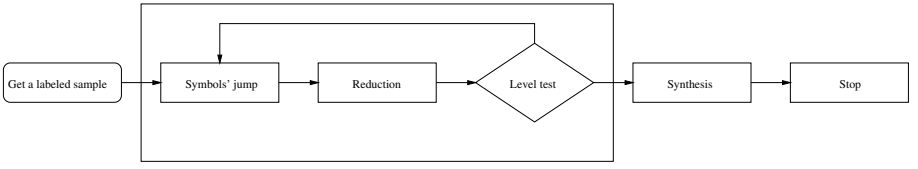



**Fig. 3.** Block diagram of PAC-meditation.

In this paper we restrict ourselves to classes of monotone boolean formulas. With $\mathfrak{X} = \mathbf{X}_n \equiv \{0,1\}^n$ we construct the atomic components of 0-level borders which call canonical monomial and clauses described by the propositional variables $V_n = \{v_1, \ldots, v_n\}$ as follows.

**Definition 1.** *i) given* $\mathbf{X}_n$ *and set* $E^+$ *of positive examples, a monotone monomial* $\mathsf{m}$ *with arguments in* $V_n$ *is a* canonical monomial *if an* $\mathbf{x} \in \mathbf{E}^+$ *exists such that for each* $i \in \{1, \ldots, n\},\ v_i \in \mathrm{set}(\mathsf{m})$ *if* $x_i = 1,\ v_i \not\in \mathrm{set}(\mathsf{m})$ *otherwise*

*ii) given* $\mathbf{X}_n$ *and set* $E^-$ *of negative examples, a monotone clause* $\mathsf{c}$ *with arguments in* $V_n$ *is a* canonical clause *if an* $\mathbf{x} \in \mathbf{E}^-$ *exists such that for each* $i \in \{1, \ldots, n\},\ v_i \in \mathrm{set}(\mathsf{c})$ *if* $x_i = 0,\ v_i \not\in \mathrm{set}(\mathsf{c})$ *otherwise*

These formulas do not constrain the final expression of $g^*$ in that any Boolean formula on the binary hypercube can be represented either through the union of monomials (DNF) or through the intersection of clauses (CNF). They just represent a set of points that necessarily must belong to $g^*$ given a positive example or can not belong to it given a negative one. Moreover, let us consider a function $\mathbf{s}$ that, in analogy to $\mathbf{S}$, assigns to a concept a set of inner sentry points sentinnelling from inside the concept w.r.t. other concepts included in it. Thus they are a minimal set of points internal to the concept $c$ to be sentinelled and external to at least one other included in it, with analogous features and functions. Our atomic formulas need only one example as inner or outer frontier.

According to the above canonical monomials are a richer representation of positive points and a more concise one as well in that, if one monomial contains another we can skip the latter from the set. Their union constitutes an inner border (the union of the thin contoured gray circles in Fig. 1) since represents a minimal hypothesis on $g^*$. Similar properties hold for the canonical clauses, whose intersection now represents the maximal hypothesis consistent with $g^*$,and then an outer border. These duties derive from the fact that they represent properties which we infer from the points after the monotonicity assumption. These properties pivot around the fact that positive examples are inner sentries for these monomials and negative examples for the clauses. Now, to render this prerogative proof against any other representation through monomials (clauses), i.e. any other consistent association of monomials to inner points, we must fix these examples as sentry points of the largest expansions of canonical monomials (narrowing of canonical clauses) which still prove consistent with negative (positive) points. This is the distinguishing feature of our abstraction process: we pass from a lower to higher level representation of partial hypotheses in such a way that new sentry points are a subset of older ones (with some points possibly becoming useless due to the expansion).

Applying the distributive property to the union of two monomials: $v_i v_j v_k \vee v_l v_m v_n v_r = (v_i \vee v_l v_m) \wedge (v_i \vee v_n v_r) \wedge (v_j v_k \vee v_l v_m) \wedge (v_j v_k \vee v_n v_r)$ we obtain a new monomial where literals are constituted by clauses and, in turn, literals in the clauses are substituted by monomials. Let us generalize the operation, keeping groups of at most $k_2$ monomials and splitting them in such a way that at most $k_1$ hyperclauses arise. Bounds on $k$'s stand for requisites of conciseness on the formula description, i.e. for a compression of our knowledge. Then we examine the case of extending (enlarging) the single atomic formulas in a consistent way. We do the same with hyperclauses. A very easy procedure for doing these tasks is reported in [10].

Cycling along the block diagram in Fig.3 and denoting $\cup^t = \cap^{t-1} = \cap$ if $t$ is odd and $/= ,0\cup$ otherwise, for $t \geq 0$, at the $L^{\text{th}}$ abstraction level we obtain formulas belonging to the families of hyper_$L$_monomials $\mathbf{G}_{n;k_0,k_1,\ldots,k_{\nu-1}}$ and hyper_$L$_clauses $\mathfrak{G}_{n;k_0,k_1,\ldots,k_{\nu-1}}$ whose elements $g$ can be written respectively as follows, for $\nu = 2L$, $k'_i \leq k_i$ for each $i < \nu$, $k'_\nu \in \mathbb{N}$ and suitable $q$

$$g = \begin{cases} \bigcup_{j_0=1}^{k'_0} {}^{1} \bigcup_{j_1=1}^{k'_1} {}^{2} \cdots \bigcup_{j_\nu=1}^{k'_\nu} {}^{\nu+1} v_{q(j_0,j_1,\ldots,j_\nu)} & \text{for the former, and} \\ \bigcup_{j_0=1}^{k'_0} {}^{0} \bigcup_{j_1=1}^{k'_1} {}^{1} \cdots \bigcup_{j_\nu=1}^{k'_\nu} {}^{\nu} v_{q(j_0,j_1,\ldots,j_\nu)} & \text{for the latter} \end{cases}$$

In short, we pass from one level to the next adding a pair of operations of the "$\cap\cup$" kind for hypermonomials and "$\cup\cap$" for hyperclauses. When we are satisfied with the achieved abstraction level, we abandon the loop and try to synthesize the two borders in a unique formula in the *Synthesis* block, meeting possible further requirements. Obviously, not each formula may comply with the actual borders, since they come from a somehow biased gap reduction. We denote *mininside* and *maxinside* the classes $\mathbf{C}_m(r; L)$ and $\mathbf{C}_M(r; L)$ of formulas given by the disjunction of at most $r$ hyper_$L$_monomials and the conjunction of at most $r$ hyper_$L$_clauses respectively. These formulas are obtained from an exhaustive check on all the possible $r$-partitions of the hyperpoints constituting the inner or outer border. For $r$ growing with $n$ this constitutes a highly costly computational job, the escape from which is to learn an esier hypothesis. The complexity of a learning job indeed might strongly depend on the representation of the hypothesis. For instance, it is well known that learning $k$-term-DNF($n$) formulas is NP-hard for every preassigned $k \geq 3$ but is polynomial if werepresent them through $k$-CNF formulas (a less concise representation) [11]. We speak of *proper learning* when concept and hypothesis classes coincide. In our framework, we can decide either *proper learning* the final formulas by activating *Synthesis* or *non proper learning* it by precisely relying on the current inner or outer borders. Since getting the accuracy targets $\varepsilon$ and $\delta$ as in Lemma 1 requires in any case a polynomial number of examples we have:

**Lemma 2.** *[10] At every fixed abstraction level, PAC-meditation algorithm supplies in polynomial time inner and outer borders as non proper hypotheses for the target concept with accuracy parameters $\varepsilon$ and $\delta$. Learning mininside or maxinside classes is an* NP*-easy problem.*

Under sentinels management perspective we start associating an atom (monomial or clause) to each example. Then we reduce the number of atoms checking inclusion relations either during the symbolic jump or after the reduction of the formulas. We still have monomials and clauses, each needing a unique sentinel. Iteration of the abstraction module leads similarly to further sentinels reductions. Namely, at the first level each monomial is a sentinel of the 1-level hyperformulas. But since a hypermonomial for instance comes from the union of more than one monomial it may need more than one hyperpoint for its sentineling. The point we stress is that our symbolic representation is such that the hyperformula will be sentineled by the same number of examples, whatever the abstraction level we use for representing the sentinels.

**Definition 2.** *Given a concept class $\mathbf{G}_n$ on $\mathbf{X}_n$, we say that $\mathbf{G}_n$ is* learnable without information waste up to level $L$ *from its borders if there exists an algorithm that for any example set $E = E^+ \cup E^-$ produces consistent hypotheses $h \in \mathbf{G}_n$ whose borders at level $L$ are sentineled by a same subset of $E$, whatever the level $i \leq L$ of the abstraction at which they are represented.*

**Theorem 1.** *[10] PAC-meditation learns* mininside $\mathbf{C}_m(r; L)$ *and* maxinside $\mathbf{C}_M(r; L)$ *without information waste up to level $L$ whenever* Synthesis *finds a solution.*

## 4 Numerical Results and Conclusions

We have applied the PAC-meditation algorithm in many case studies and real world problems. Concerning the formers a $k$_term-DNF formulas are learnt at the first abstraction level as they belong to $\mathbf{G}_{n;a}$ for $a$(ny) number of literals per term. The complexity of the problem, NP-hard indeed, descends from the necessity of reducing to k the number of monomials originally raising from the positive examples.

We treated with our approach the problem of learning the symbolic representation of some emotional states starting from the speach of some people involved in a talk show. Reports on the solution can be find at the web site http://www.image.ntua.gr/physta. The table below is a typical report of a learning session allowing us to state logical necessary and sufficient conditions for characterizing the emotion sadness at level 1, where symbols $a$ to $f$ are linked to symbols $v_1$ to $v_{24}$ through relations such as $a = v_{23}v_{24}$, $b = v_{15}+v_{22}$, $c = v_{15} + v_{13}$.

| Border | 1-level formulas |
|---|---|
| Inner | $ab + cde + def$ |
| Outer | $d * (c + b + f) * e$ |

Splitting the learning process in a sequence of two sided converging approximations appears an efficient approach typical of the human brain. The procedure we propose can be easily extended in two directions. More specific constraints can be stated for the set-union and set-intersection operations at the basis of the abstraction jumps, to embed other kinds of formal knowledge in addition to the bounds on the hyperterm complexity. In addition, we might consider a lot of relaxed meditation schemes, based for instance on neural network and fuzzy sets paradigms.

## References

[1] Valiant, L.G.: A theory of the learnable. Comm. of the ACM **11** (1984) 1134–1142
[2] Blum, A.: Learning boolean functions in an infinite attribute space. Machine Learning **9** (1992) 373–386
[3] Schapire, R.E.: The strength of weak learnability. Machine Learning **2** (1990) 197–227
[4] Linial, N., Mansour, Y., Rivest, R.L.: Results on learnability and the vapnik-chervonenkis dimension. Information and Computation **90** (1991) 33–49
[5] Qunilan, J.R.: C4.5: programs for machine learning. Morgan Kaufmann Publishers, San Mateo, California (1993)
[6] Andrews, R., Geva, S.: Inserting and extracting knowledge from constrained backpropagation network. In: Proc. 6th Australian Conference on Neural Networks, Sidney (1995) 29–32
[7] Vapnik, V.: The Nature of Statistical Learning Theory. Springer, New York (1995)
[8] Apolloni, B., Malchiodi, D., Orovas, C., Palmas, G.: From synapses to rules. Cognitive Systems Research (2002) in press.
[9] Apolloni, B., Chiaravalli, S.: Pac learning of concept classes through the boundaries of their items. Theoretical Computer Science **172** (1997) 91–120
[10] Apolloni, B., Baraghini, F., Palmas, G.: PAC meditation on boolean formulas. Technical report, Università degli Studi di Milano (2001)
[11] Pitt, L., Valiant, L.: Computational limitations on learning from examples. J. ACM **35** (1988) 965–984

# On the Reformulation of Vehicle Routing Problems and Scheduling Problems*

J. Christopher Beck[1], Patrick Prosser[2], and Evgeny Selensky[2]

[1] 4C, University College Cork, Ireland cbeck@4c.ucc.ie
[2] Department of Computing Science, University of Glasgow, Scotland.
pat/evgeny@dcs.gla.ac.uk

**Abstract.** We can reformulate a vehicle routing problem (VRP) as an open shop scheduling problem (SSP) by representing visits as activities, vehicles as resources on the factory floor, and travel as set up costs between activities. In this paper we present two reformulations: from VRP to open shop, and the inverse, from SSP to VRP. Not surprisingly, VRP technology performs poorly on reformulated SSP's, as does scheduling technology on reformulated VRP's. We present a pre-processing transformation that "compresses" the VRP, transforming an element of travel into the duration of the visits. The compressed VRP's are then reformulated as scheduling problem, to determine if it is primarily distance in the VRP that causes scheduling technology to degrade on the reformulated problem. This is a step towards understanding the features of a problem that make it more amenable to one technology rather than another.

## 1 Introduction

In the capacitated vehicle routing problem with time windows, $m$ identical vehicles initially located at a depot are to deliver discrete quantities of goods to $n$ customers. Each customer has a demand for goods and each vehicle has a capacity. A vehicle can make only one tour starting at the depot, visiting a subset of customers, and returning to the depot. Time windows define an interval for each customer within which the visit must be made. A solution is a set of tours for a subset of vehicles such that all customers are served only once and time window and capacity constraints are respected. The objective is to minimise distance travelled, and sometimes additionally to reduce the number of vehicles used. The problem is NP-hard [2].

An $n \times m$ job shop scheduling problem (JSSP) consists of $n$ jobs and $m$ resources. Each job consists of a set of $m$ completely ordered activities. Each activity requires a resource and has an execution duration on that resource. The complete ordering defines a set of precedence constraints, such that an activity cannot begin execution until the preceeding activity has completed. Activities that require the same resource cannot overlap in their execution and no pre-emption is allowed. The problem is then to decide if all activities can be

* This work was supported by EPSRC research grant GR/M90641 and ILOG SA.

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 282–289, 2002.

scheduled within a given makespan, while respecting the resource and precedence constraints. The job shop scheduling decision problem is NP-complete [2].

What are the characteristics of vehicle routing problems that make them more amenable to local search techniques allied to construction methods? What properties of scheduling problems make them more suitable to systematic search and powerful constraint propagation? In this paper, we take a first step toward answering these questions. We present reformulations between the VRP and factory shop scheduling problems[1] (SSP's), and identify three significant differences between VRP's and SSP's. We then propose a pre-processing transformation of VRP's that reduces one of these differences, namely the ratio of processing times to transition times. We then investigate the behaviour of this transformation using standard VRP and JSSP benchmarks.

## 2 Transformations: VRP $\leftrightarrow$ SSP

**VRP $\rightarrow$ SSP:** We reformulate the VRP into a SSP as follows. Each vehicle is represented as a resource, and each customer visit as an activity. The distance between a pair of visits corresponds to a transition time between respective activities. Each activity can be performed on any resource, and is constrained to start execution within the time window defined in the original VRP. Each activity has a demand for a secondary resource, corresponding to a visit's demand within a vehicle. For each resource $R$ there are two special activities $Start_R$ and $End_R$. Activities $Start_R$ and $End_R$ must be performed on resource $R$. $Start_R$ must be the first activity performed on $R$ and $End_R$ the last. The transition time between $Start_R$ and any other activity $A_i$ corresponds to the distance between the depot and the $i^{th}$ visit. Similarly the transition time between $End_R$ and $A_i$ corresponds to the distance between the depot and the $i^{th}$ visit. The processing time of $Start_R$ and $End_R$ is zero. We associate a consumable secondary resource with every (primary) resource to model the capacity of vehicles. Consequently a sequence of activities on a resource corresponds to a vehicle's tour in the VRP. In the resultant SSP each job consists of only one activity, each activity can be performed on any resource, and there are transition times between each pair of activities. The problem is then to minimise the sum of transition times on all machines and maybe also to minimise the number of resources used.

**SSP $\rightarrow$ VRP:** We have for each resource a vehicle, and for each activity a customer visit. The visits have a duration the same as that of the corresponding activities. Each visit can be made only by the vehicles corresponding to the set of resources for the activity. Any ordering between activities in a job results in precedence constraints between visits. Transition times between activities correspond to travel distances between visits. The deadline $D$ imposes time windows on visits. Assuming we have $m$ resources, and therefore $m$ vehicles, we have $2m$ dummy visits corresponding to the departing and returning visits to the depot.

[1] We will refer to the jobshop and the open shop scheduling problems as the shop scheduling problem, i.e. SSP.

A vehicle's tour corresponds to a schedule on a resource. For the $n \times m$ JSSP we have a VRP with $m(n+2)$ visits and $m$ vehicles. Each visit can be performed only by one vehicle. Since there are no transition times in the JSSP, there are no travel distances between visits, but visits have durations corresponding to those of the activities. There are precedence constraints between those visits corresponding to activities in a job. The decision problem is then to find an ordering of visits on vehicles that respects the precedence constraints and time windows.

In [3] Selensky investigated the two extreme cases identified above, i.e. the reformulation of benchmark VRP's as SSP's (and their subsequent solution using ILOG Scheduler) and the reformulation of JSSP benchmarks as VRP's (solved using ILOG Dispatcher). The VRP's used were the 56 Solomon benchmarks [4] and jobshop problems of size 6×6 [6] and 15×15 [5]. The VRP instances reformulate to strange open shop problems, having essentially single activity jobs, a vast selection of resources for each activity, vanishingly small durations, and comparatively enormous transition times. Conversely the JSSP instances map to VRP's where a visit can be performed only by one vehicle and there is no travel! Not surprisingly, the performance of the tools was greatly degraded as the problems were reformulated.

While these results do not come as a great surprise, they do lend support to the belief that there are characteristics of vehicle routing and scheduling problems that make them more suitable to their standard resolution technology. Three differences stand out. The transformed VRP have many more alternative resources than typical scheduling problems. The SSP has more complex temporal relations than the typical VRP. The activity duration is small and transition is large in VRP, whereas in SSP it is the converse. In this paper we focus on just one of these, the ratio of activity duration to transition time. We present a transformation that attempts to reduce the difference between the transition time and activity duration in VRP's and investigate its effect on problem solving.

## 3 Compressing the VRP

The basic idea of this transformation is to reduce travel within a VRP by adding a portion of travel costs to the costs of visits. Consequently, when the VRP is reformulated as a SSP that SSP will have activities with relatively large durations and small set up cost and look more like a *normal* SSP than a VRP. First, we show how to compress a travelling salesman problem (TSP). We then take into consideration time windows, and show how to apply our transformation to the VRP.

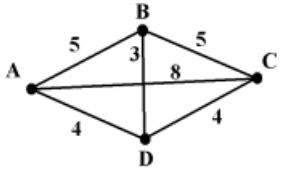
**Compressing the TSP:** Consider a travelling salesman problem where nodes have costs as well as edges. The cost of visiting a node is then the cost of the edge entering the node plus the cost of the edge exiting the node, plus the cost of the node itself. We can transform this cost such that the node cost is increased and the costs on the entering and exiting edges are reduced.

Consider node $j$, with entering edge $(i, j)$ and exiting edge $(j, k)$, with costs $C_j$, $C_{i,j}$ and $C_{j,k}$ respectively. Let $C_{min} = min(C_{i,j}, C_{j,k})$. We can reduce the

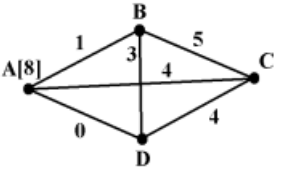
cost of both edges by $C_{min}$, such that one of these becomes zero, and add to the node cost $2 \times C_{min}$. This preserves the cost of entering, visiting, and exiting $j$. More generally, for a TSP we can process each node $i$ as follows:

1. let $C_{min}$ be the cost of the cheapest edge incident on node $i$;
2. for each edge incident on node $i$ subtract $C_{min}$ from the edge's cost;
3. add a cost of $2 \times C_{min}$ to $C_i$, the cost of node $i$;
4. the node now has at least one incident edge of zero cost.

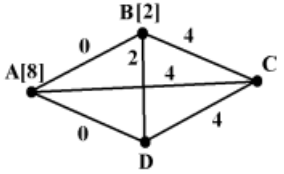
We need only process each node once, i.e. after processing, a node has at least one zero cost incident edge and re-processing will have no effect. Figure 1 shows the sequence of transformations of a four-node clique. All nodes start with zero cost and are processed in alphabetic order. Note that if we processed the vertices in a different order we might end up with a different final graph, i.e. the transformation is order dependent.



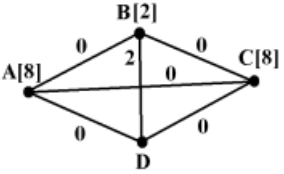

(a) Original graph




(b) Step 1. Processing node $A$




(c) Step 2. Processing node $B$




(d) Step 3. Processing node $C$

**Fig. 1.** Transformation of a 4-node clique. New node costs are shown in square brackets.

**Compressing the VRP:** In a Hamiltonian path we have two distinguished nodes, i.e. the start node $s$ and end node $e$, and the transformation is then modified as follows. Since $s$ is not entered and $e$ is not exited, we do not add to those vertices twice the cost of its minimum incident edge. Instead, we add the minimum cost once only, and process all other vertices as above.

When time windows are associated with nodes the compression can change the lower and upper bounds on the time of entering and exiting a node. Consider the problem of finding a Hamiltonian path from $s$ to $e$, where each node $i$ has a time window $[ES_i, LS_i]$, i.e. we must arrive at node $i$ no earlier than $ES_i$ and no later than $LS_i$. Assume we have a graph with edges $(s, i)$, $(s, j)$, and $(i, j)$ where $C_{s,i} < C_{s,j}$ and $C_{i,j} < C_{s,j} - C_{s,i}$. Further assume that we process nodes

in the order $s$, then $i$, then $j$. On processing start node $s$, the transformed costs of node $s$ becomes $C'_s = C_s + C_{s,i}$, and transformed edges $C'_{s,j} = C_{s,j} - C_{s,i}$, and $C'_{s,i} = 0$. Node $i$ is then processed, and this has no effect since it has a zero cost incident edge $C'_{s,j}$. On processing node $j$ we get the transformed costs $C'_j = C_j + 2 \times C_{i,j}$, $C''_{s,j} = C'_{s,j} - C_{i,j}$, and $C'_{i,j} = 0$. Note that via substitution $C''_{s,j} = C_{s,j} - C_{s,i} - C_{i,j}$. Consequently, on the transformed graph the cost of travelling directly from $s$ to $j$ is $C_s + C_{s,j} - C_{i,j}$, i.e. we arrive earlier on the transformed graph and might now have to wait before entering node $j$. A symmetric argument holds for leaving the node.

**Theorem 1.** *On transforming an arbitrary node $i$, with time window $[ES_i, LS_i]$, its earliest start time becomes $ES_i - C_{i,j}$ and its latest start time becomes $LS_i - C_{i,j}$, where $C_{i,j}$ is the cost of the cheapest edge incident on node $i$.*

**Proof.** We need to prove that on travelling from $i$ to $j$, or from $j$ to $i$ in the transformed graph we maintain any slack that existed in the original graph. The proof is by cases and is presented in full in [1]. *QED*

**Order Dependence:** The transformation is order dependent. We will investigate three transformations. The first, and most obvious, uses a lexicographic ordering of vertices. We will refer to this as a *lex* ordering. The second, we call *maxMin*; when selecting a node $i$ to process next we choose a node such that its cheapest incident edge is a maximum. For example, in Figure 1 this would initially select node A or node C, as their cheapest incident edges are largest, with a cost of 4. The intuition behind this is that it will attempt to make the biggest reduction in edge costs. The third ordering is *minMin*. This might be thought of as the anti-heuristic, selecting to process a node $i$ with smallest minimum incident edge cost. In Figure 1 this would initially choose node B or D.

## 4 An Empirical Study

After the VRP has been compressed, we can then reformulate the VRP as a SSP and solve it using scheduling technology. We now attempt to determine if any of these compressions of VRP benchmarks result in better solutions, with or without the reformulation to a SSP.

The experiments were performed on the 56 Solomon benchmarks [4]. These problems fall into six classes: C1, C2, R1, R2, RC1, and RC2. All problems in a class have the same set of customer visits, i.e. there is one set of 100 x/y coordinates corresponding to the customer visits. What differentiates problems within a class is the distribution of time windows and demands. In C1 and C2 visits are clustered, and in C2 visits have larger time windows and increased vehicle capacity, i.e. C2 is a less constrained set than C1. The locations of visits in R1 and R2 are random, and again, each R2 instance has larger time windows and increased vehicle capacities. The RC set is made up of clustered groups of randomly generated visits, and again RC2 is a less constrained set than RC1.

The 56 benchmark VRP were transformed as above, using the lex ordering, maxMin ordering, and minMin ordering. This gives us a total 224 problems, i.e. the original problems and each problem transformed using the three orderings. The problems were then solved using ILOG Dispatcher, each instance being allowed 10 minutes cpu time. The purpose of this was to determine how VRP solving technology is influenced by the amount of travel within the problem. That is, will Dispatcher perform worse or better as we compress problems, transforming travel between nodes into the cost of processing those nodes? The same set of 224 problems were then reformulated as scheduling problems and solved using ILOG Scheduler, and again given 10 minutes cpu time per instance. Would the transformation result in an improvement in Scheduler's performance on the reformulated VRP's?

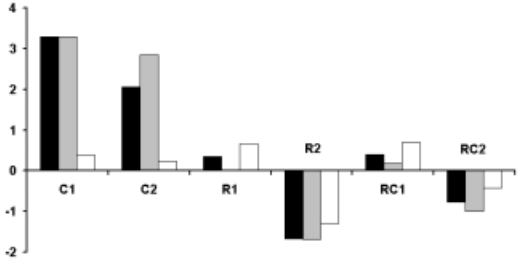



**Fig. 2.** Percentage increase in cost as a result of transforming VRP's and solving with Dispatcher

**Solving with Dispatcher:** Figure 2 shows the percentage change in cost as a result of the transformation when solving the VRPs with Dispatcher, i.e. without reformulation. The black bars represent lex ordering, the grey bars maxMin ordering, and minMin ordering is in white. Results are expressed as the average percentage change in cost compared to the original problem solved with Dispatcher. We can see that for clustered problems all of the orderings result in an increase in cost with a maximum of over 3%. An analysis of the first solutions found, prior to improvement with local search, showed that the first solutions were not considerably affected by the transformations (the largest increase in cost of the first solutions being less than 0.3%). Unfortunately, the results for problems with a random element in customer distributions are not so clear. However, this still demonstrates the sensitivity of the VRP solving technology to the mix of travel and processing time. The maxMin ordering attempts to make the largest compression of the problem, and this corresponds to our worst performance of Dispatcher. This tends to suggest that the VRP technology degrades as the VRP becomes more like a SSP. This might also suggest that an inverse transformation, one that was able to *stretch* the VRP by converting the cost of processing a visit into additional travel, might improve the VRP technology.

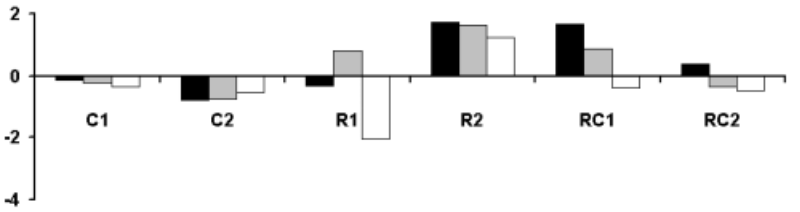



**Fig. 3.** Percentage increase in cost as a result of transforming VRP's, reformulating them, and solving with Scheduler

**Solving with Scheduler:** Each of the 224 problems were then reformulated as SSP and solved using ILOG Scheduler. In Figure 3 we see again the percentage change in cost as a result of the transformation (again with black bars for lex ordering, grey for maxMin ordering, and white for minMin ordering). In the clustered problems, C1 and C2, we see a reduction in cost, and a comparatively larger improvement in the relaxed C2 problems. In the random problems, R1 and R2, there tends to be an increase in cost, even greater in the less constrained R2 problems. This might suggest that the transformations do not fare well in unstructured problems. In the RC problems it appears that the transformations reduce cost but only when the problems are less constrained (i.e. the RC2 problems).

Distance appears to be a crucial factor when solving VRP's with VRP technology. When reformulated as scheduling problems, transition times (i.e. travel time) have more of an impact when problems have structure.

## 5 Conclusion and Future Work

We have presented reformulations between VRP's and SSP's. By solving the reformulated problems with domain specific technologies we have demonstrated that the vehicle routing technology appears to be well suited to VRP's and not at all suited to reformulated SSP's [3]. The converse also appears to hold. However we must add the caveat, that this is no surprise because the benchmark problems used are extreme cases and should indeed be well fitted to their solving technologies.

We have presented a transformation process for the VRP, which can be used as a pre-processing step before solving a problem, or reformulating and solving a problem. This transformation attempts to *compress* the VRP by adding an element of travel into the processing of each node. This was done in the hope that the reformulated problem would appear to be more like a SSP than a VRP, i.e. duration of activities would be increased and transition times would be decreased. Our experiments showed that the transformations can degrade

the VRP technology, suggesting that indeed travel is an important problem feature. When reformulated as SSP it was less clear. Although the activities now have increased duration and decreased transition times, they are still peculiar problems: they remain as single activity jobs that can be performed on any resource.

In our ongoing studies we are producing VRP's with more structure, and structure that we can control. This is done by gradually varying time windows, vehicle capacity, heterogeneity of the fleet, sequencing constraints between visits, etc. By gradually increasing the richness of these VRP's we expect to reach a point where SSP technology competes with VRP technology. Our study will attempt to determine just what features bring about this competition. In addition, we plan to investigate the inverse transformation i.e. rather than compress a VRP we might stretch it. Might this be a pre-processing step that will improve VRP solving? Also, when there are set up costs in the SSP, might a further compression improve solving?

Could the transformations and reformulations be of any real use? We believe so. As we add more constraints to the VRP, such as sequencing constraints between visits, restrict time windows, and specialise visits to a subset of the fleet, VRP's will tend to be more like SSP's. Similarly, as the tooling on the shop floor becomes more flexible, such that machines can perform many functions, and the cost of re-configuring tools increases, the SSP will also tend to have features similar to the VRP. Therefore on a spectrum that has the VRP at one end and the JSSP at the other, we expect that as problems become richer it will be less clear as to just what technology is most appropriate. At that time we might expect that transformations and reformulations will become an important tool.

**Acknowledgements.** This work was done while the first author was employed by ILOG, SA.

## References

1. J. Christopher Beck, Patrick Prosser, and Evgeny Selensky. On the reformulation of vehicle routing problems and scheduling problems. Technical Report APES-44-2002, APES Research Group, February 2002. Available from http://www.dcs.st-and.ac.uk/ apes/apesreports.html.
2. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.
3. Evgeny Selensky. On mutual reformulation of shop scheduling and vehicle routing. In *Proceedings of the 20th UK PLANSIG*, pages 282–291, 2001.
4. M. Solomon. Algorithms for the Vehicle Routing and Scheduling Problem with Time Window Constraints. *Operations Research*, 35:254–365, 1987.
5. Jssp benchmarks, 15 by 15. http://www.dcs.gla.ac.uk/pras/resources.html.
6. Jssp benchmarks, 6 by 6. http://www.ms.ic.ac.uk/jeb/pub/jobshop1.txt.

# The Oracular Constraints Method

T.K. Satish Kumar[1] and Richard Dearden[2]

[1] Knowledge Systems Laboratory, Stanford University
[2] Research Institute for Advanced Computer Science / NASA Ames Research Center
tksk@ksl.stanford.edu, dearden@ptolemy.arc.nasa.gov

**Abstract.** Constraint satisfaction and combinatorial optimization form the crux of many AI problems. In constraint satisfaction, feasibility-reasoning mechanisms are used to prune the search space, while optimality-reasoning is used for combinatorial optimization. Many AI tasks related to diagnosis, trajectory tracking and planning can be formulated as hybrid problems containing both satisfaction and optimization components, and can greatly benefit from a proper blend of these independently powerful techniques.
We introduce the notion of model counting to bridge the gap between feasibility- and optimality-reasoning. The optimization part of a problem then becomes a search for the right set of constraints that must be satisfied in any good solution. These constraints, which we call the *oracular constraints*, replace the optimization component of a problem to revive the power of constraint reasoning systems.

## 1 Introduction and Motivation

Constraint satisfaction and combinatorial optimization form the crux of many AI problems. In constraint satisfaction, feasibility-reasoning mechanisms are used to prune the search space, while optimality-reasoning is used for combinatorial optimization. Sometimes reformulating one to the other helps us to tackle hybrid problems containing both a satisfaction and an optimization component. A good example of this is the fractional packing problem [7]. Here, oracles are defined for a potentially hard problem by replacing a satisfaction component of the problem with the minimization of a certain potential function. Solving a series of these oracles leads us to a solution for the original problem. An instance of the fractional packing problem is the multi-commodity flow problem [6] that is solved using repeated calls to the minimum cost circulation problem (which acts as an oracle).

In this paper, we examine the converse idea of reformulating the optimization component of a problem using oracular constraints. While constraint reasoning has witnessed the development of very sophisticated techniques for solving a number of AI problems, none of these techniques can be extended in a straightforward manner when the problem involves an optimization component as well. For example, there are many systems (Graph-Plan [1], Black-Box [3] etc.) that solve planning problems involving complex and cascading levels of constraint reasoning, but adding an optimization component to the planning problem by associating costs with actions makes them non-amenable to these techniques.

Reformulating optimization as satisfaction allows us to cast a hybrid problem in satisfaction and optimization as a series of pure constraint satisfaction problems that

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 290–298, 2002.

serve as oracles for the original problem. The oracles are solved making use of the full power of constraint reasoning systems. The number of calls we make to the oracles can be traded off against closeness to the optimal solution for the original problem. In many cases, this continuum is very important and can be exploited to do real-time problem solving since it is often reasonable to find a near-optimal solution rather than trying to solve the potentially much harder problem of computing the optimal solution.

Oracular constraints can also serve as a compact representation for all the reasonably good solutions to a combinatorial optimization problem. A good example of why this can be important is found in trajectory tracking [4]. Here, the best trajectories given the current observations may not be the best once future observations are available. Ideally therefore, we would like to track all the reasonable hypotheses rather than just the best ones (in case the best hypotheses now are ruled out by subsequent observations). The problem is that the number of reasonable candidates can increase over time beyond our computational resources and it becomes impossible to track all of them explicitly. Reformulating the optimization component (posterior probabilities conditioned on observations) associated with the trajectories as satisfaction, we can represent all the good hypotheses at any given point in time as solutions to these oracular constraints. By tracking these constraints rather than the candidates themselves, our representation scales only polynomially with time.

## 2 Approach

Consider a hybrid constraint satisfaction-optimization problem $H$. Let the set of variables in the system be $V$ and let us assume that each variable $v_i$ can take only one of a finite set of values defined as its domain ($dom(v_i)$). We are faced with the task of finding an assignment for all the variables in $V$ such that some set of constraints are satisfied and some metric is optimized. We make use of the following definitions:

**Definition** *(Hybrid Problem in SAT and OPT)* A problem $H$ is said to be hybrid if it has a satisfaction component $H_{SAT}$ of the form "Find $X \in W$" and an optimization component $H_{OPT}$ of the form "optimize $cost(X)$". $H$ is then characterized by $(H_{SAT}, H_{OPT}) = (W, cost)$.

**Definition** *(Optimization Family)* An *optimization family* consists of two parts: (1) a subset of variables $S_i = \{X_{i_1}, X_{i_2} \cdots X_{i_{j(i)}}\}$, called the family ($S_i \subseteq V$) (2) a function $opt_i$ defined over $S_i : D_{i_1} \times D_{i_2} \cdots D_{i_{j(i)}} \rightarrow R$ (the reals).

An optimization family defined on some subset of the variables can be thought of as an independent component of the global objective function, which is formed by applying some composition operator (see below) to the optimization families. We assume that the values of the function $opt_i$ are normalized i.e. $opt_i$ maps an assignment of values to variables in $S_i$ to a real number in [0, 1], called the *value* of that assignment. We will assume that there are a total of $C$ optimization families in the system.

**Definition** *(Value of a Complete Assignment)* The *value of a complete assignment* to all variables in $V$ is a combination of the values defined by each optimization family in the system, using a predefined composition operator ($\times$).

In most cases, the composition operator is multiplication. In other cases, it might be addition or the minimum. For illustration and simplicity, we will assume that this

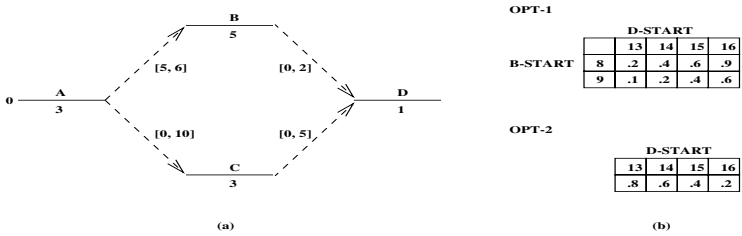



**Fig. 1.** A hybrid scheduling problem. (a) shows four actions (solid lines) and the constraints (dashed arrows) between them. An arrow labeled $[a, b]$ between two actions means that the second must be executed at least $a$ and at most $b$ seconds after the first. (b) shows $H_{OPT}$ in the form of two optimization families.

operator is multiplication. It is easy to verify that the theory developed in this paper can be generalized to most other useful operators.

For example, consider the hybrid scheduling problem shown in Figure 1(a). There are four actions each with a fixed duration, and a number of constraints relating the execution times of the actions (assuming integer starting times for all actions). The problem's optimization component consists of the two optimization families shown in Figure 1(b). OPT-1 prefers plans that maximize the time between actions B and D, while OPT-2 prefers shorter plans overall.

The general approach we will take in this paper is to first convert the numbers provided in the optimization families to a binary representation. The bits constituting these binary representations are used to cast constraints that progressively capture the optimization component of the hybrid problem. The system casts a series of pure constraint satisfaction problems incorporating $H_{SAT}$ and a set of additional constraints acting as oracles to the optimization part of the problem. Depending upon whether or not a solution is found to the previous problem, the system casts a new problem using these bits and a generic control procedure called the *bargain heuristic*. By doing this, we hope to establish a continuum for trading off the number of times we solve oracles with the proximity of our solution to the optimal one.

## 3 Optimization and Model Counting

We now present an interesting relationship between optimization and model counting. The model counting problem is the problem of counting the number of solutions to a satisfiability problem (SAT) or a constraint satisfaction problem (CSP).

**Definition** *(Binary Representation of an OPF)* The *binary representation of an OPF* is a table in which all the floating-point entries of the OPF are re-written in a binary form up to a precision of $P$ binary digits and the decimal point along with any redundant zeroes to the left of it are removed.

We provide a set of definitions and results relating the value of a partial assignment $A$ to the number of solutions (under the same partial assignment $A$) to CSPs composed out of the binary representations of the OPFs. Basic definitions related to CSPs can be found in [2]. We refer to the set of all optimization families as the network.

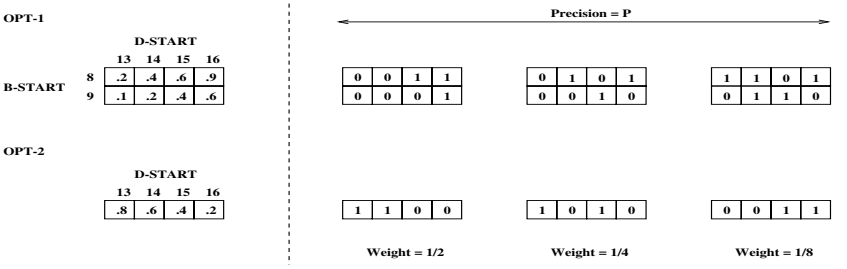



**Fig. 2.** The optimization families (OPFs) from our example are written in a columnar form on the left. On the right are their binary representations, one bit per matrix. For example, the top left entry of OPT-1 is 0.2, or 0.001 in binary (to three digits).

**Definition** *(Zero-one-layer of an OPF)* The $k^{th}$ *zero-one-layer of an OPF* is a table of zeroes and ones derived from the $k^{th}$ bit position of all the numbers in the binary representation of that OPF.
**Definition** *(Weight of a zero-one-layer)* The $k^{th}$ zero-one-layer of an OPF is defined to have weight $2^{-k}$.
**Definition** *(CSP Compilation of an OPF)* The $k^{th}$ *CSP compilation of an OPF* is a constraint over the variables of the OPF that is derived from the $k^{th}$ zero-one-layer of the OPF such that zeroes correspond to disallowed tuples and ones correspond to allowed tuples. We will write $h_{ik}$ for the $k^{th}$ CSP compilation of the $i^{th}$ OPF.
**Definition** *(CSP Compilation of Network)* The $(k_1, k_2 \cdots k_C)$ *CSP compilation of the network* is the set of constraints $S = \{s_i : s_i$ is the $k_i^{th}$ CSP compilation of the $i^{th}$ OPF$\}$. We write this as $CSP_{(k_1,k_2\cdots k_C)}$.
**Definition** *(Weight of a CSP Compilation)* The *weight of a* $(k_1, k_2 \cdots k_C)$ *CSP compilation* of a network is defined to be equal to $2^{-(k_1+k_2\cdots k_C)}$.

The left side of Figure 2 shows the optimization families from Figure 1. On the right are the binary representation of the OPFs to a precision of three bits. The top right matrix in the figure is the third zero-one-layer of the first OPF, and has weight $1/8$. The $(3, 1)$ CSP compilation of the network is a set of constraints including the third matrix in the top row, and the first in the other row, and has weight $1/16$.
**Property** The total number of CSP compilations possible is $P^C$.
**Notation** Let $A$ indicate a complete assignment to the variables. We write $CSP_{(k_1,k_2\cdots k_C)}(A)$ to indicate whether $A$ satisfies all the constraints of $CSP_{(k_1,k_2\cdots k_C)}$.
**Theorem 1** The value of a complete assignment $A = (X_1 = x_1, X_2 = x_2 \cdots X_n = x_n)$ is the sum of the weights of the CSP compilations of the network that are satisfied by the assignment. That is, $P(A) = \sum_{(k_1,k_2\cdots k_C)} CSP_{(k_1,k_2\cdots k_C)}(A)2^{-(k_1+k_2\cdots k_C)}$ (for all $1 \leq i \leq C, 1 \leq k_i \leq P$).

In our example, any schedule where **D** starts at time 15 and **B** at time 8 satisfies $CSP_{(1,2)}$ and $CSP_{(1,3)}$ and has value $1/8 + 1/16$. This approximates the true value of the assignment (0.24), based on CSP compilations and according to Theorem 3.

We note in passing that the number of optimization families in the system and hence the number of CSP compilations in the model counting reformulation can be reduced by clustering together optimization families [2]. A trade-off can be made between the number of OPFs and the size of each one.

## 4 Oracular Constraints

In hybrid problems, we are often interested in a complete assignment—that is, an assignment of values to all the variables in the system. For example, complete plans optimized to certain costs rather than having any notion of marginalizing over variables. This can be exploited to reverse engineer the search for good solutions as a search for the constraints that these solutions must satisfy.

**Theorem 2** Let $H_i$ denote $\{h_{ij} | 1 \leq j \leq P\}$. If a complete assignment satisfies $S_i \subseteq H_i$ (for $1 \leq i \leq C$), then the value of that assignment is $\Pi_{i=1}^{C} \sum_{j=1}^{P} 2^{-j}(h_{ij} \in S_i)$.

**Theorem 3** If the optimization problem is solved up to $t$ bits of precision, then the solution found is within $C \times 2^{-t}$ of the optimal solution.

The optimization metrics in hybrid problems usually come from probabilities or costs defined on actions etc. In general these problems are of the form: "Find $X$ such that $X \in W$ and $cost(X)$ is optimized". For example, the problem of finding cheapest plans when costs are associated with actions has $W$ equal to the set of all valid plans and the cost function equal to the sum of the costs associated with the actions included in the plan. The problem of trajectory tracking also falls into this framework with costs being defined as the posterior probabilities conditioned on the observations.

In many of these problems, finding valid candidates in the space $W$ is itself hard. Constraint reasoning systems (like Black-Box or Graph-Plan) that work well for finding such valid candidates generally do not also optimize the cost. The idea is to revive such systems by including additional constraints such that only those candidates that have a reasonably optimized value of the cost satisfy them. These additional oracular constraints are derived from the optimization families of the system (like conditional probability tables etc.). Note that traditional methods for handling hybrid problems like branch and bound, constraint propagation etc. are not expected to perform well when $H_{SAT}$ is very hard. In such cases, $H_{SAT}$ is best solved using local search techniques, stochasticity, heuristics, random restarts etc. which cannot work in conjunction with procedures like branch and bound. The addition of oracular constraints allows us to use these incomplete search techniques which are well-suited for solving $H_{SAT}$.

**Definition** *(Candidate Generating System)* A *candidate generating system* $W(H)$ for a hybrid problem $H = (H_{SAT}, H_{OPT})$ is a feasibility reasoning system that solves the satisfaction component of the problem $H_{SAT}$ augmented with some other constraints $H_{ORC}$ (see below).

Typically, $H_{ORC}$ is "small" compared to $H_{SAT}$ and we assume that $W(H)$ can easily incorporate $H_{ORC}$ if it can solve $H_{SAT}$ by doing potentially complex and cascading levels of constraint reasoning.

**Definition** *(Oracle, Oracular Constraints and Oracular Method)* An *oracle* $R(H) = (H_{SAT}, H_{ORC})$ to the hybrid problem $H = (H_{SAT}, H_{OPT})$ is the problem of finding $X$ that satisfies $H_{SAT}$ and a set of *oracular constraints* $H_{ORC}$ such that repeated calls

to a candidate generating system $W(H)$ that can solve $H_{SAT} \cup H_{ORC}$ (with potentially different $H_{ORC}$ in different iterations) converge to a solution for $H$. A control strategy that does this is called an *oracular method*.

**Input:** A hybrid problem $H = (H_{SAT}, H_{OPT})$.
**Output:** An assignment $X$ such that $H_{SAT}(X)$ and $H_{OPT}(X)$ is "good".
**Initialize:** Set the iteration to 0, BestSolution to NULL.
**While** the iteration $\leq$ some precision **do**
Construct $H_{ORC}$ based on $H_{OPT}$ and the solution to the previous oracle.
Let $X$ be the solution to the oracle $R(H) = (H_{SAT}, H_{ORC})$ (using $W(H)$).
Let BestSolution be the best of BestSolution and $X$.
**Return** BestSolution.

**Fig. 3.** A generalized Oracular Constraints algorithm.

The intuition behind the oracle is that the optimization component of $H$ ($H_{OPT}$) is replaced by a satisfaction component in $R(H)$ ($H_{ORC}$). By solving a sequence of oracles with different oracular constraints, we approximate the optimal solution to the original hybrid problem. Figure 3 shows the general form of an oracular method.

**Definition** *(Monotonic)* An oracular method $M(H)$ is *monotonic* if and only if the worst candidate solution produced by an oracular call at iteration $i$ ($X_i$) is strictly better (in terms of the optimization value) than the worst candidate solution produced by an oracular call at iteration $j$ ($X_j$) when $i > j$.

The monotonic property is of significance in characterizing oracular methods. A monotonic oracular method ensures us that we are always making progress in improving the quality of the solutions. This continuum can be exploited to do real-time problem solving where it is acceptable to produce near-optimal solutions within a window of opportunity rather than finding the optimal solution (which could be much harder).

**Definition** *(Completeness and Safety)* An oracular method $M(H)$ is *complete* if all optimal solutions to the original problem are eventually reported by it (in a number of iterations that depends on the precision of the numbers used), and is *safe* if all candidates reported by it are indeed optimal.

### 4.1 The Bargain Heuristic

The bargain heuristic is essentially a walking strategy on the constraints resulting out of a model counting reformulation of the optimization component of a problem. The basic idea is to exploit the structure of the weights on the constraints. Consider a maximization problem in which we have just one optimization family $F$ and we have it broken up into constraints $F_1, F_2 \cdots F_P$ (as usual, $P$ is the precision of the numbers used). Any candidate solution to $H_{SAT}$ that also satisfies $F_1$ must have a value of at least 1/2. If there is such a candidate solution, we can constrain the problem even more by adding in $F_2$. Any solution to $\{H_{SAT}, F_1, F_2\}$ must have value at least 3/4 and so on. If at any point there is no solution, then we replace the constraint added most recently (say $F_i$) by OR-ing it with the next one ($F_i \bigvee F_{i+1}$). This method clearly works to produce the

optimal candidate (assuming that we are trying to maximize) because of the structure of the weights on the constraints. A candidate satisfying $F_1$ is better than one which does not satisfy $F_1$ even if the latter satisfies any number of $F_i$ ($i > 1$) and so on. When oracles are unsolvable, we use OR-ing rather than simply removing constraints and trying new ones, to ensure that backtracking is never required.

In the case of multiple optimization families, the above strategy can be used as a heuristic. Given the decomposition of all optimization families $OPF_i$ to $H_i = \{h_{ij} | 1 \leq j \leq P\}$, we start out by assuming that there is a solution to all the constraints with weight 1/2 ($h_{i1}$ for all $i$). If this oracle has a solution, then we greedily add (see below) another constraint (breaking ties arbitrarily). If at any point there is no solution to the current set of constraints, we relax some constraints making greedy choices again. The relaxation phase is done not by retracting constraints but by OR-ing the most recently added constraint with its successor (as above). We will see that this strategy achieves both computational tractability and the monotonic property.

**Input** A hybrid problem $H = (H_{SAT}, H_{OPT})$
**Output** A Boolean formula $bform$ that all solutions to $H$ (for precision $P$) satisfy. The clauses in $bform$ consist of the constraints that make up each optimization family. The formula will be constructed by adding ANDs and ORs into the slots between the constraints within each family, with ANDs between the families.
**Initialize:** Set all the slots to be empty.
**While** there are unfilled slots and the desired precision hasn't been reached **Do**
Let $bform$ be the formula defined by the currently filled slots.
**If** $bform \cup H_{SAT}$ has a solution, **then**
Add an AND to the leftmost unused slot (breaking ties randomly).
**Else**
Add an OR to the slot to the right of the most recently filled slot.
**Return** $bform$.

**Fig. 4.** The bargain heuristic algorithm for constructing a formula that all good solutions to a hybrid problem must satisfy.

Using the convention that ORs have a precedence over ANDs in Boolean evaluation, the goal is to place ANDs and ORs in the $(P-1) \times C$ slots available between $h_{ij}$ and $h_{i(j+1)}$ for $1 \leq i \leq C, 1 \leq j < P$. The bargain heuristic progressively introduces ANDs and ORs in these slots to converge to a Boolean formula that captures the optimization part of the problem. The heuristic is guided by the results of previous calls to the oracle. A pointer is maintained to the first unused slot in each row so that ANDs and ORs can be introduced without backtracking. We note that the only place where negation occurs in the resulting Boolean formula is in the zero-one-layers of the optimization families. The algorithm is shown in Figure 4.

**Theorem 4** The number of iterations taken by the bargain heuristic using a precision P is $C \times (P-1)$ (where $C$ is the number of optimization families).

**Theorem 5** *(Monotonicity)* The bargain heuristic is a monotonic oracular method.

**Theorem 6** *(Goodness)* Any solution produced by the bargain heuristic has a value $\geq \prod_{i=1}^{C} \sum_{j=1}^{P-1} 2^{-j}(Slots[i][j] == AND)$. Here, $Slots[i][j]$ is the slot between $h_{ij}$ and $h_{i(j+1)}$.

The bargain heuristic essentially tries to minimize $j$ (and thereby maximize the contributing factor $2^{-j}$) by introducing ANDs as soon as possible (by adding an AND in the leftmost unused slot whenever the current oracle is solvable).

### 4.2 Further Applications of Oracular Constraints

We now briefly discuss another important application of oracular constraints, representing the set of "good" solutions to a problem. This is useful in situations where data arrives online, for example in trajectory tracking. Here the goal is to diagnose a complete or partial assignment to the variables of a system given the observations made as those variables evolve over time. The challenge is to do this diagnosis incrementally (reusing computation made for earlier diagnosis calls) as and when new observations are made. The fundamental problem for trajectory tracking is that because the data comes online, the candidate hypotheses that are best suited for the current observations may not turn out to be the best when the future observations are available. One solution is therefore to track many hypotheses all of which are reasonably good, in the hope that some of them can be extended to suit the future observations. However, at some intermediate points in time, the number of candidate hypotheses that are reasonably good may become exponentially large (in the number of time steps) and we cannot track all of them using a polynomial amount of space. The problem is therefore not entirely computational, but is also one of representation. By tracking individual candidates, we are not making use of the fact that many hypotheses share the same values for some variables.

Tracking oracular constraints instead of individual hypotheses allows us to represent all the good hypotheses as the set of solutions to these constraints (the number of which scales only polynomially with time). For example, in the case of a single optimization family, the first zero-one-layer represents all candidates which have a value at least 0.5 etc. We can also define levels of "goodness" corresponding to different levels of precision. The number of levels of goodness however is bounded by a user defined constant (rather than being exponential) and the representation remains polynomial.

## 5 Conclusions

We have presented an approach for solving hybrid problems with a hard satisfaction component by using a model counting reformulation of the optimization component. This provides an alternative to recent methods that use constraint propagation to derive better lower bounds for guiding branch and bound search [5] etc. We expect our approach to be effective on a different class of problems than these approaches, specifically problems where the $H_{SAT}$ part of the problem is relatively hard (like in planning and diagnosis) and complete search procedures such as branch and bound are unlikely to find solutions. In contrast, if the $H_{SAT}$ part of the problem is simple or non-existent, our approach is unlikely to offer any computational gain. We are currently conducting experiments to test the efficacy of our approach for a variety of problems.

Proofs of the Theorems are in the full version of this paper, available from the authors.

## References

1. Blum, A. and Furst, M. 1997. Fast Planning Through Planning Graph Analysis. Artificial Intelligence 90, 281-300.
2. Dechter, R. 1992. Constraint Networks. Encyclopedia of Artificial Intelligence, second edition, Wiley and Sons, pp 276-285, 1992.
3. Kautz, H., McAllester, D. and Selman, B. 1996. Encoding Plans in Propositional Logic. Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning, 374-384.
4. Kurien, J. and Nayak, P. P. 2000. Back to the Future for Consistency-Based Trajectory Tracking. Proceedings of the 17th National Conference on Artificial Intelligence (AAAI 2000).
5. Larrossa, J. and Meseguer, P. Partition-based Lower Bound for Max-CSP. 1999. Proceedings of CP'99, pages 303–315.
6. Leighton, T., Makedon, F., Plotkin, S., Stein, C., Tardos, E. and Tragoudas, S. 1991. Fast Approximation Algorithms for Multicommodity Flow Problem. In Proceedings of the 23rd ACM Symposium on the Theory of Computing, pages 101-111, May 1991.
7. Plotkin, S., Shmoys, D. and Tardos, E. 1995. Fast Approximation Algorithms for Fractional Packing and Covering Problems. Math of Oper. Research, 20(2):257-301, 1995.

# Performance of Lookahead Control Policies in the Face of Abstractions and Approximations

Ilya Levner, Vadim Bulitko, Omid Madani, and Russell Greiner

University of Alberta, Edmonton AB, Canada T6G 2E8,
{ilya|bulitko|madani|greiner}@cs.ualberta.ca

**Abstract.** This paper explores the formulation of image interpretation as a Markov Decision Process (MDP) problem, highlighting the important assumptions in the MDP formulation. Furthermore state abstraction, value function and action approximations as well as lookahead search are presented as necessary solution methodologies. We view the task of image interpretation as a dynamic control problem where the optimal vision operator is selected responsively based on the problem solving state at hand. The control policy, therefore, maps problem-solving states to operators in an attempt to minimize the total problem-solving time while reliably interpreting the image. Real world domains, like that of image interpretation, usually have incredibly large state spaces which require methods of abstraction in order to be manageable by today's information processing systems. In addition an optimal value function ($V^*$) used to evaluate state quality is also generally unavailable requiring approximations to be used in conjunction with state abstraction. Therefore, the performance of the system is directly related to the types of abstractions and approximations present.

## 1 Introduction

Image interpretation is a complex and adaptive task. It is complex in that there is rarely a one-step mapping from image data to scene interpretation; instead, a series of transformations is required to bridge the gap between sensory input and scene description. Examples of these transformations include region segmentation and labelling, texture filtering, and the construction of 3D depth maps, each having a multitude of parameter settings. The task is adaptive in the sense that there is no fixed sequence of actions that will recognize all objects in all settings. For example, the steps required to locate and identify trees in winter are different from the steps required to find trees in summer. In general, the most effective action sequence may depend on many features of a specific image, including lighting conditions, point of view, and the type, shape, and relative position of objects present. We therefore view this task as a dynamic control problem where the optimal vision operator is selected based on the problem solving state at hand. Thus the control policy maps problem-solving states to operators in an attempt to minimize the total problem-solving time while reliably interpreting the image.

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 299–307, 2002.

This study is motivated by the Forest Inventory Mapping System (FIMS) [Bulitko et al. 2000] currently under development. Having detailed inventories of forest resources is of tremendous importance to forest industries, governments, and researchers. Such a system would aid planning wood logging (planting and cutting), surveying against illegal activities, and ecosystem and wildlife research. Given the dynamic nature of forest evolution, the task of forest mapping is a continuous undertaking, with the objective of re-mapping the estimated 344 million hectares of Canadian forests on a 10-20 year cycle. Remote-sensing based approaches appear to be the only feasible solution to inventorizing the estimated $10^{11}$ trees. Despite numerous previous attempts [Gougeon 1993, Larsen, Rudemo 1997,Pollock 1994], no robust forest mapping system exists to date.

Recent research [Draper et al. 2000] has shown that adaptive control policies computed by assuming an underlying Markov decision process (MDP) can outperform hand coded control mechanisms on real-world computer vision tasks. The MDP framework allows for many potential benefits in addressing the control problems in image interpretation effectively, as this framework permits general control models and has a number of successful solution methodologies. In this paper, we present the FIMS system (section 2), which treats object recognition as a Markov decision process (MDP) control task. Furthermore, we introduce state abstraction and action approximation and demonstrate their benefits and shortcomings. Section 3 presents the test bed domain used in our experiments. Finally a lookahead control policy is described and analyzed that operates within the abstracted MDP space to enable efficient application of vision operators within the FIMS domain (section 4).

## 2 Image Interpretation in the FIMS Domain

In the complete FIMS system, image interpretation is composed of several layers of operators (Figure 1). [Bulitko et al. 2000] explains the benefits of a multi-layer processing approach in meeting the challenges of the forestry domain. At each stage or layer, there are two or more competing operators: several feature extractors, segmenters, labellers, canopy groupers, reconstructors, and renderers. The final labelling contains information such as the location, type, size, age, and other attributes for each tree identified within the image. The system is then able to perform 3D-reconstruction and rendering, enabling FIMS to compute the similarity between the initial image and the rendered image. This allows the intermediate layers as well as the final scene interpretation to be evaluated on their accuracy of interpretation. In each step the control policy (agent) chooses among applicable operators by *selectively* envisioning their effects several plies ahead. It then evaluates the states forecasted using its *heuristic value function,* $\tilde{V}^*$ and selects the operator leading to the most promising state

In the FIMS system, a *sequence* of operators is required in order to produce a 3D interpretation from raw images, and they have to be selected *dynamically* based on the state observed. By viewing the operators as actions and their respective data inputs and outputs as states we can formulate the task of image

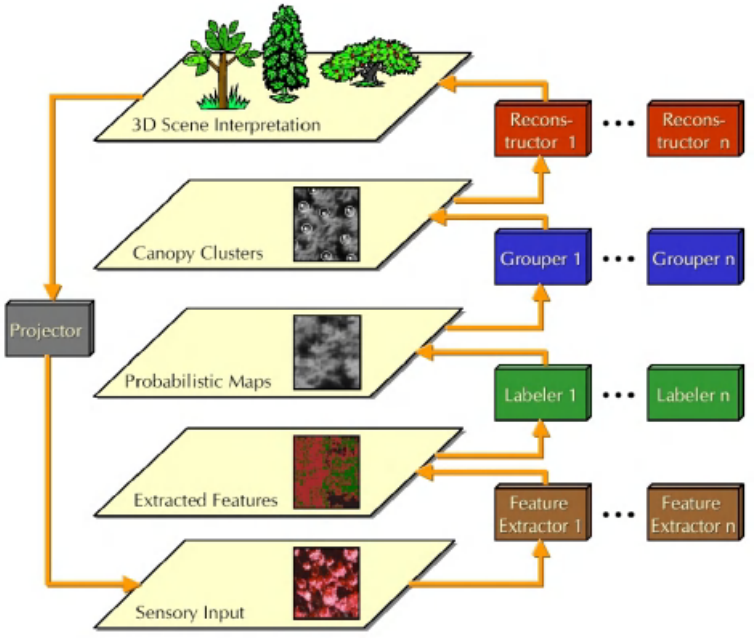



**Fig. 1.** The layers (phases) of image interpretation in FIMS.

interpretation as a Markov Decision Process [Sutton, Barto 2000]. Therefore, FIMS can be viewed as an extension of the classical MDP framework along the following directions: (i) feature functions for state space reduction, (ii) approximation of value function ($\widetilde{V}^*$) and domain model ($\widetilde{\delta}$) via machine learning methods, and (iii) explicit accuracy/efficiency tradeoff for performance evaluation.

### 2.1 Indeterminate Goal States

In FIMS, the goal state contains a geo-referenced 3D forest map with the minimum possible amount of error. Unfortunately, such a state is impossible to recognize since the sought forest maps are typically unknown. The lack of recognizable goal states means that we cannot use standard goal-searching algorithms such as A*, IDA*, RBFS, etc. [Korf 1990]. As mentioned previously, the renderer enables the 3D scene description to be transformed back into pixel level data thereby allowing for comparison between the input and final output. Currently this is the only way to determine how close the derived image interpretation matches the underlying pixel data.

### 2.2 Partial State Observability

Raw states are often enormous in size – FIMS requires on the order of $10^7$ bytes to describe a *single* problem-solving state and furthermore the state space is composed of billions of these megabyte sized states. Thus, the raw state space is infeasible to handle. A common remedy is to employ a *state abstraction function* $\mathcal{F}$ that maps large raw states to smaller abstracted states specified via extracted feature values. ADORE [Draper et al. 2000] uses feature functions to reduce multi-megabyte raw images to a handful of real-valued parameters such as the average image brightness or edge curvature. As a result, the control policy operates over the abstracted state space usually recasting the entire problem as a Partially Observable Markov Decision Process (POMDP) [Sutton, Barto 2000].

### 2.3 Value Function Inaccuracies

In order to help select an operator to apply we use a value function that maps each problem-solving state to a numeric score, thereby setting a preference relation over the set of reachable states. It is typically defined as the reward the system would get from that state on by acting optimally. If the control policy had access to the actual optimal value function $V^*$, [Sutton, Barto 2000] it would be able to act optimally in a simple greedy fashion, i.e. take actions leading to states ($s$) with the highest expected $V^*(s)$ score. Unfortunately, as the true value function is usually unavailable, we use an approximation $\widetilde{V}^*$ (denoted with a tilde). Various inaccuracies in the approximate $V^*$ often lead to sub-optimal action choices and force back-tracking, as noted in [Draper et al. 2000].

Two main sources of inaccuracies are present in the $\widetilde{V}^*$. The first source is state abstraction via the use of feature extraction function $\mathcal{F}(s)$, which can map several distinct states to one abstracted state or tile. Formally, there may be distinct states $s_i, s_j$ within our state space $S$, such that $s_i, s_j \in S$ & $s_i \not= s_j$ & $\mathcal{F}(s_i) = \mathcal{F}(s_j)$. Therefore, $\widetilde{V}^*(\mathcal{F}(s_i)) = \widetilde{V}^*(\mathcal{F}(s_j))$. The second source of errors is noise due to machine learning. Since the optimal value function is being approximated via machine learning techniques, errors are inevitable. Even if the feature function did not combine two distinct raw states into one abstracted state, due to machine learning the two distinct abstracted states may still be mapped to the same value, ie $\widetilde{V}^*(\mathcal{F}(s_i)) = \widetilde{V}^*(\mathcal{F}(s_j))$, while $\mathcal{F}(s_i) \not= \mathcal{F}(s_j)$.

To offset the value function inaccuracies we employ a lookahead control policy, to envision future states. The motivation for using search is derived from the domain of games such as chess [Hsu et al. 1995] and checkers [Schaeffer et al. 1992] where search is the method of choice in overcoming inaccuracies present in the state valuation function. Unlike the games domain, where one would like to search as deep as possible, the FIMS domain has further inaccuracies (described next) prohibiting the use of deep lookahead.

### 2.4 Inaccurate Domain Models

Individual computer vision operators can take hours to compute on large images. Combined with the exponential number of operator applications needed

for the lookahead long running times make the actual operators unusable for envisionment. Thus, approximate versions of such operators comprising the domain model $\widetilde{\delta}$ are employed within the lookahead. Consequently, such simplified models are inaccurate and, therefore, unable to foresee future states precisely. Sequential noisy predictions introduce compounded errors into envisioned states and thus into $\widetilde{V}^*$ values which offset the benefits of deep lookahead searches.

### 2.5 Solution Strategy

FIMS clearly exemplifies the typical problems commonly encountered in real world domains. The need for state abstraction to reduce the state space, the existence of value functions inaccuracies, and errors in the state transition function prevent the applicability of classical MDP and search methodologies. However, a fine tuned combination of feature functions, state evaluation functions, domain model approximations combined with dynamic lookahead depth selection may in fact produce near optimal results. Naturally these problems motivate an investigation into the conditions under which the use of a lookahead control policy is most applicable.

## 3 Experimental Domain

Many Reinforcement Learning projects have used the grid world domain as a test bed [Sutton, Barto 2000]. In the following we refine the classical definition by introducing state abstraction in a particular fashion compatible with the real-world FIMS domain. This compatibility enables further scalability studies via a transition from the grid world to the actual FIMS system. The refined grid world is referred to as the Maze Domain. The *maze* is represented by an $N \times N$ two-dimensional matrix $M$ with each cell being in two possible states: *empty* $M(x, y) = 0$ or *wall* $M(x, y) = 1$. There is a single *agent* in the maze that can occupy any of the empty cells. Additionally, one of the empty cells $(x_g, y_g)$ contains a *goal*. The maze is surrounded by a solid wall preventing the agent from wandering off the map. An *agent's raw state* is a pair of coordinates $(x, y)$ with $0 \leqslant x, y \leqslant N - 1$. Formally: $S = \{0 \ldots N - 1\} \times \{0 \ldots N - 1\}$.

The set $A$ of agent's actions is comprised of $\lambda$ equally spaced directions and a special action 'quit'. Each of the $\lambda$ move actions transports the agent along a ray shot from the agent's current location at the angle of $\frac{360 \cdot a}{\lambda}$ degrees where $0 \leq a < \lambda$. The Euclidean distance travelled is upper-bounded by $\tau$ and walls encountered (Figure 2). We represent this with a deterministic state transition function $\delta_d(s, a) = s'$.

The MDP *immediate reward* function is defined as:

$$r(s, a, s') = \begin{cases} \mathcal{R}(s'), & \text{if } a = \text{'quit'}, \\ -\|s, s'\|, & \text{otherwise.} \end{cases} \tag{1}$$

Here the cost $\|s, s'\| = \sqrt{(x_s - x_{s'})^2 + (y_s - y_{s'})^2}$ of action $a$ is defined as the Euclidian distance between the new and the old cells. The terminal reward $\mathcal{R}(s)$

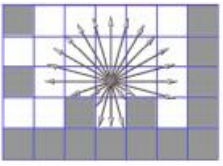


**Fig. 2.** Agent's actions in the maze domain ($\lambda = 24, \tau = 2$). Cells occupied by walls are inaccessible (shown in grey).

of quitting in state $s$ is defined as: $\mathcal{R}(s) = \Theta - E_{sp}(s, s_g)$ ,where $\Theta$ is a constant, and $E_{sp}(s, s_g)$ is the sum of Euclidean distances of steps in the shortest path from state $s$ to $s_g$. A game terminates whenever the agent executes the 'quit' action or the move quota is exhausted.

### 3.1 State Abstraction

Within the maze domain, the state abstraction is simulated via *fixed tiling*. Specifically, if the raw state is $s = (x, y)$ the abstracted state is given as:

$$\mathcal{F}_k(x, y) = \left( \left\lfloor \frac{x}{k} \right\rfloor \cdot k, \left\lfloor \frac{y}{k} \right\rfloor \cdot k \right) \tag{2}$$

where the floor function $\lfloor\ \rfloor$ returns the integer part of its argument. Parameter $k = 1, 2, \ldots, N$ is the *degree of abstraction*. Effectively, the entire maze is divided into non-overlapping rectangular $k \times k$ tiles.

The underlying motivation of this particular abstraction scheme is compatibility with state abstraction in FIMS. Namely, in a computer vision system like FIMS and ADORE, most operators add an information datum of a different category to the problem-solving state (e.g., add extracted edges to an image). The state abstraction features used for various types of data vary considerably (e.g., average brightness for images and mean curvature for extracted edges) making it unlikely that an operator application leaves the agent in the same abstraction tile. On the other hand, several similar operators can move the agent to a single tile (different from the current one). Furthermore, some computer vision operators are not applicable in certain problem-solving states which is modelled by certain angles being blocked off by the maze walls.

## 4 Lookahead Control Policy

If state aggregation has the property that a longer sequence of actions is more likely to lead outside an abstracted state, then deeper lookahead plays an important beneficial role. By considering only one action or a short sequence of actions the agent can observe that all reachable states have the same values. Thus in such a case deeper lookahead increases the probability of the agent "seeing" outside the current state abstraction tile and seems to be quite beneficial. Therefore, lookahead can provide the agent with a more accurate guidance

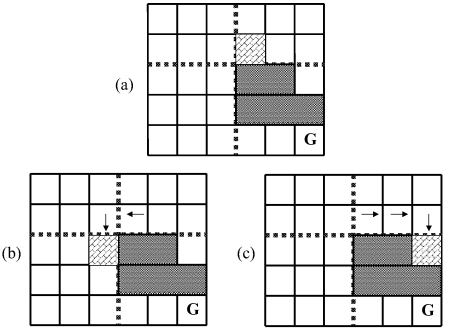



**Fig. 3. (a)** A maze domain partitioned into four state abstraction tiles (dashed lines). The agent's starting position is the mesh filled cell and the goal position is marked with a "G". Walls are shown as grayed out, solid cells. For this example there are 4 actions (north, east, south, west) available to the agent (i.e. $\lambda = 4$). **(b)** Final position of the agent using lookahead with depth of $p = 1$. **(c)**Final position of the agent using lookahead with a depth of $p = 3$.

function. Is deeper lookahead *always* beneficial? Figure 3, illustrates an example of adverse effects of a deeper lookahead in a simple maze domain. There are 5 deterministic actions: up, down, left, right, and 'quit' available to the agent. The agent has perfect knowledge of its position within the maze but the value function is the same for all states in each abstracted state (or tile). Assume that the agent starts in the mesh filled square of Figure 3a. A *greedy* policy considers only a lookahead of 1 ply deep. Choosing an action under such policy leads the agent to the state denoted by the mesh filled square in Figure 3b (which is a local maximum provided depth 1 lookahead). But if the agent considers sequences of 3 actions (a lookahead of 3 plies deep) it ends up in the state denoted by a mesh filled square of Figure 3c. This cell is a much worse state than the terminal state in Figure3b. Thus deeper lookahead results in a policy leading the agent away from the goal rather than bringing it closer. This example illustrates adverse impact of abstracting together states with very different $V^*$ values. In general, however, we found that deeper lookahead does help. The results consistently demonstrated that deeper lookahead in the face of state abstraction is beneficial with the aforementioned exceptions being rare. Figure 4 demonstrates the agents average performance within a larger $(48 \times 48)$ maze domain with $\lambda = 8$ and a distanse of each action $\tau = 2\sqrt{2}$. Clearly lookahead plays a beneficial role in this case by helping the agent 'see' beyond the abstraction tile it currently occupies.

In contrast to the above results, we found cases where lookahead did neither improve nor degrade the agents performance within the maze domain. Figure 5 shows the performance results of an agent using a machine learned $\widetilde{V}^*$ function. Unlike errors due to state abstraction, errors induced by machine learning do not appear to be diminished by deeper lookahead search.

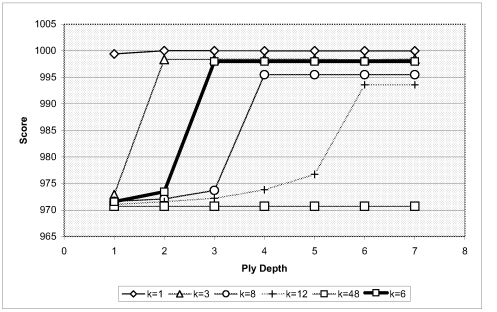



**Fig. 4.** Reward (score) vs. Ply Depth for experiments with various state abstraction tile sizes ($k$). Clearly lookahead improves the agent's performance, with optimal ply depth (resulting in maximal reward) proportionally rising with the increase in state abstraction, $k$.

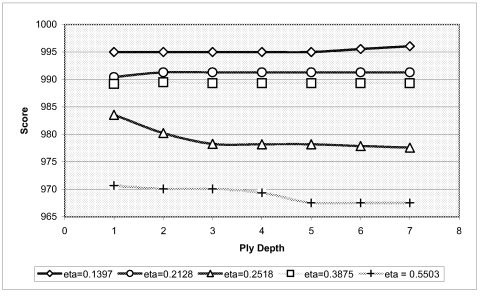



**Fig. 5.** Reward (score) vs. Lookahead Ply Depth for experiments using machine learned $\widetilde{V}^*$ function. $\eta$ is the measure of error present within the $\widetilde{V}^*$ with respect to $V^*$. It is clear there are cases of $\widetilde{V}^*$ where lookahead does neither help or hinder the performance of an agent.

## 5 Summary and Future Work

This preliminary study focused on identifying the benefits of lookahead control policies within MDP/POMDP environments. Clearly, even for the trivial cases presented, lookahead can have significantly different effects on the agent's performance with respect to the rewards gained. Such contrasting results motivate further studies to establish concrete rules for when the benefits of lookahead outweigh the computational costs of doing so. Furthermore as seen in experiments presented even without the erroneous state transition (domain) model the benefits of lookahead drop off at specific ply depths depending on the types of inaccuracies present. It is therefore beneficial to establish a concrete testing procedure for determining optimal ply depth within the lookahead control policies.

**Acknowledgements.** We would like to thank Natural Sciences and Engineering Research Council (NSERC) for their sponsorship of this project.

## References

[Bulitko et al. 2000] Bulitko, V., Caelli, T., McNabb, D. 2000. Forestry Information Management System (FIMS): An Introduction. Technical Report. University of Alberta.

[Draper et al. 2000] Draper, B., Bins, J., Baek, K. 2000. ADORE: Adaptive Object Recognition, *Videre*, 1(4):86–99.

[Good 1971] Good, I.J. 1971. Twenty-seven Principles of Rationality. In *Foundations of Statistical Inference*, Godambe V.P., Sprott, D.A. (editors), Holt, Rinehart and Winston, Toronton.

[Gougeon 1993] Gougeon, F.A. 1993. Individual Tree Identification from High Resolution MEIS Images, *In Proceedings of the International Forum on Airborne Multispectral Scanning for Forestry and Mapping*, Leckie, D.G., and Gillis, M.D. (editors), pp. 117-128.

[Hsu et al. 1995] Hsu, F.H., Campbell, M.S., Hoane, A.J.J. 1995. Deep Blue System Overview. *Proceedings of the 9th ACM Int. Conf. on Supercomputing*, pp. 240-244.

[Korf 1990] Korf, R.E. 1990. Real-time heuristic search. *Artificial Intelligence*, Vol. 42, No. 2-3, pp. 189-211.

[Larsen, Rudemo 1997] Larsen, M. and Rudemo, M. 1997. Using ray-traced templates to find individual trees in aerial photos. *In Proceedings of the 10th Scandinavian Conference on Image Analysis*, volume 2, pages 1007-1014.

[Newborn 1997] Newborn, M. 1997. *Kasparov vs. Deep Blue: Computer Chess Comes Out of Age*. Springer-Verlag.

[Pollock 1994] Pollock, R.J. 1994. A Model-based Approach to Automatically Locating Tree Crowns in High Spatial Resolution Images. *Image and Signal Processing for Remote Sensing*. Jacky Desachy, editor.

[Schaeffer et al. 1992] Schaeffer, J., Culberson, J., Treloar, N., Knight, B., Lu, P., Szafron, D. 1992. A World Championship Caliber Checkers Program. *Artificial Intelligence*, Volume 53, Number 2-3, pages 273-290.

[Sutton, Barto 2000] Sutton, R.S., Barto, A.G., 2000. *Reinforcement Learning: An Introduction*. MIT Press.

# TTree: Tree-Based State Generalization with Temporally Abstract Actions*

William T.B. Uther and Manuela M. Veloso

Computer Science Department,
Carnegie Mellon University,
Pittsburgh, PA 15213 USA
{uther, veloso}@cs.cmu.edu

**Abstract.** In this paper we describe the Trajectory Tree, or TTree, algorithm. TTree uses a small set of supplied policies to help solve a Semi-Markov Decision Problem (SMDP). The algorithm uses a learned tree based discretization of the state space as an abstract state description and both user supplied and auto-generated policies as temporally abstract actions. It uses a generative model of the world to sample the transition function for the abstract SMDP defined by those state and temporal abstractions, and then finds a policy for that abstract SMDP. This policy for the abstract SMDP can then be mapped back to a policy for the base SMDP, solving the supplied problem. In this paper we present the TTree algorithm and give empirical comparisons to other SMDP algorithms showing its effectiveness.

## 1 Introduction

Both Markov Decision Processes (MDPs) and Semi-Markov Decision Processes (SMDPs), presented in [1], are important formalisms for agent control. They are used for describing the state dynamics and reward structure in stochastic domains and can be processed to find a policy; a function from the world state to the action that should be performed in that state. In particular, it is useful to have the policy that maximizes the sum of rewards over time. Unfortunately, the number of states that need to be considered when finding a policy is exponential in the number of dimensions that describe the state space. This exponential state explosion is a well known difficulty when finding policies for large SMDPs.

A number of techniques have been used to help overcome exponential state explosion and solve large (S)MDPs. These techniques can be broken into two main classes. *State abstraction* refers to the technique of grouping many states together and treating them as one abstract state, e.g. [2,3,4]. *Temporal abstraction* refers to techniques that group sequences of actions together and treat them as one abstract action [5,6,7]. Using a function approximator for the value function, e.g. [8], can, in theory, subsume both state and temporal abstraction, but the authors are unaware of any of these techniques that, in practice, achieve significant temporal abstraction.

* This research was sponsored by the United States Air Force under Agreement Nos. F30602-00-2-0549 and F30602-98-2-0135. The content of this publication does not necessarily reflect the position of the funding agencies and no official endorsement should be inferred.

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 308–315, 2002.

In this paper we introduce the Trajectory Tree, or TTree, algorithm with two advantages over previous algorithms. It can both learn an abstract state representation and use temporal abstraction to improve problem solving speed. It also uses a new format for defining temporal abstractions that relaxes a major requirement of previous formats – it does not require a termination criterion as part of the abstract action.

Starting with a set of user supplied abstract actions, TTree first generates some additional abstract actions from the base level actions of the domain. TTree then alternates between learning a tree based discretization of the state space and learning a policy for an abstract SMDP using the tree as an abstract state representation. In this paper we give a description of the behavior of the algorithm. Moreover we present empirical results showing TTree is an effective anytime algorithm.

## 2 Definitions

An SMDP is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, P, R \rangle$. $\mathcal{S}$ is the set of world states. We will use $s$ to refer to particular states, e.g. $\{s, s'\} \in \mathcal{S}$. We also assume that the states embed into an $n$-dimensional space: $\mathcal{S} \equiv \mathcal{S}^1 \times \mathcal{S}^2 \times \mathcal{S}^3 \times \cdots \times \mathcal{S}^n$. In this paper we assume that each dimension, $\mathcal{S}^i$, is discrete. $\mathcal{A}$ is the set of actions. We will use $a$ to refer to particular actions, e.g. $\{a_0, a_1\} \in \mathcal{A}$. Defined for each state action pair, $P_{s,a}(s', t) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \Re \rightarrow [0, 1]$ is a joint probability distribution over both next-states and time taken. It is this distribution over the time taken for a transition that separates an SMDP from an MDP. $R(s, a) : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ defines the expected reward for performing an action in a state.[1]

The agent interacts with the world as follows. The agent knows the current state: the world is Markovian and fully observable. It then performs an action. That action takes a length of time to move the agent to a new state, the time and resulting state determined by $P$. The agent gets reward for the transition determined by $R$. The agent now knows the new state and acts again, etc.

Our goal is to learn a policy, $\pi : \mathcal{S} \rightarrow \mathcal{A}$, that maps from states to actions. In particular we want the policy, $\pi^*$, that maximizes a sum of rewards. To keep this sum of rewards bounded, we will introduce a multiplicative discount factor, $\gamma \in (0, 1)$. The goal is to find a policy that maximizes $\sum_{i=0}^{\infty} \gamma^{\tau_i} r_i$ where $\tau_i$ is the time that the agent starts its $i^{\text{th}}$ action, and $r_i$ is the reward our agent receives for its $i^{\text{th}}$ action.

We can then define the following standard functions:

$$Q(s, a) = R(s, a) + \sum_{s' \in \mathcal{S}} \int_{t=0}^{\infty} P_{s,a}(s', t) \gamma^t V(s') \, \mathrm{d}t \tag{1}$$

$$V(s) = Q\left(s, \pi(s)\right) \tag{2}$$

$$\pi^*(s) = \underset{a \in A}{\operatorname{argmax}} \, Q^*(s, a) \tag{3}$$

[1] $R$ can also depend upon both next state and time for the transition, but as these in turn depend only upon the state and action, they fall out of the expectation.

In addition, we will define the following $T$ function. This function is defined over a set of states $\mathcal{S}' \subset \mathcal{S}$. It measures the discounted sum of reward for following the given action until the agent leaves $\mathcal{S}'$, then following the optimal policy.

$$T_{\mathcal{S}'}(s,a) = R(s,a) + \tag{4}$$

$$\sum_{s' \in \mathcal{S}'} \int_{t=0}^{\infty} P_{s,a}(s',t)\gamma^t T_{\mathcal{S}'}(s',a)\,\mathrm{d}t + \tag{5}$$

$$\sum_{s' \in (\mathcal{S}-\mathcal{S}')} \int_{t=0}^{\infty} P_{s,a}(s',t)\gamma^t V(s')\,\mathrm{d}t \tag{6}$$

We will write $T(s,a)$ instead of $T_{\mathcal{S}'}(s,a)$ when $\mathcal{S}'$ is the set of states that embed into the abstract state containing $s$.

We assume that instead of sampling $P$ and $R$ directly from the world, our agent instead samples from a *generative model* of the world, e.g. [9]. This is a function, $G : \mathcal{S} \times \mathcal{A} \to \mathcal{S} \times \Re \times \Re$, that takes a state and an action and returns a next state, a time and a reward for the transition. The returned values are sampled from $P$ and $R$. This model is more powerful than having the agent in the real world as the agent can sample from any state and action it chooses.

## 3 The TTree Algorithm

TTree works by building an abstract SMDP which is simpler than the original, or base, SMDP. The solution to this abstract SMDP is a fast approximation the solution to the base SMDP. The abstract SMDP is formed as follows: The states in the abstract SMDP, the *abstract states*, are formed by partitioning the states in the base SMDP; each abstract state corresponds to the set of base level states in one element of the partition. Each action in the abstract SMDP, an *abstract action*, corresponds to a policy in the base SMDP. The abstract transition and reward functions are found by sampling. We will use bold face to distinguish the abstract SMDP from the base level SMDP, e.g. $\mathbf{s}$ vs. $s$ or $\mathbf{A}$ vs. $\mathcal{A}$.

TTree uses a tree to partition the base level state space into abstract states. Each node in the tree corresponds to a region of state space with the root node corresponding to the entire space. Internal nodes divide the state space along one dimension with one child for each discrete value along that dimension. Leaf nodes correspond to abstract states; all the base level states that fall into that region of space are part of the abstract state.

Each abstract action corresponds to a base level policy. There are two ways in which these abstract actions can be obtained; they can be supplied by the user, or they can be generated by TTree. In particular, TTree generates one abstract action for each base level action, and one additional 'random' abstract action. The 'random' abstract action is a base level policy that performs a random base level action in each base level state. The other generated abstract actions are degenerate base level policies: they perform the same base level action in every base level state.

The abstract transition and reward functions are sampled by following trajectories through the base level state space. These functions are sampled separately for each

abstract state as follows. A set of base level states is sampled from the abstract state. From each of these start states, for each abstract action, the algorithm uses the generative model to sample a series trajectories through the base level states that make up the abstract state. In detail for one trajectory: Let the abstract state we are considering be the state $\mathbf{s}$. The algorithm first samples a set of base level start states, $\{s_0, s_1, \ldots, s_k\} \in \mathbf{s}$. It then gathers the set of base level policies for the abstract actions, $\{\pi_{\mathbf{a_1}}, \pi_{\mathbf{a_2}}, \ldots, \pi_{\mathbf{a_l}}\}$. For each start state, $s_i$, and policy, $\pi_{\mathbf{a_j}}$ in turn, the agent samples a series of base level states from the generative model forming a trajectory through the low level state space. As the trajectory progresses, the algorithm tracks the discounted reward for the trajectory, and the total time taken by the trajectory.

These trajectories have a number of stopping conditions. The most important is that the trajectory stops if it reaches a new abstract state. The trajectory also stops if the system detects a deterministic self-transition in the base level state, if an absorbing state is reached, or if the trajectory is over a predefined length. The end result is a tuple for each trajectory of the start state, abstract action, end base state, total discounted reward and total time: $\langle s_i, \mathbf{a_j}, s_{stop}, t, r \rangle$.

By finding the abstract state of the end state of the trajectory, $s_{stop} \in \mathbf{s}_{stop}$, we turn the trajectory into a sample transition in the abstract state: $\langle \mathbf{s}, \mathbf{a_j}, \mathbf{s}_{stop}, t, r \rangle$. The sample transitions are combined to estimate the abstract transition and reward functions, $\mathbf{P}$ and $\mathbf{R}$.

The algorithm now has a complete abstract SMDP. It can solve this using traditional techniques to find a policy for the abstract SMDP: a function from abstract states to the abstract action that should be performed in that abstract state. However, the abstract actions are base level policies, and the abstract states are sets of base level states, so we also have a function from base level states to base level actions; we have a policy for the base SMDP.

Having found a policy, TTree then looks to improve the accuracy of its approximation by increasing the resolution of the state abstraction. It does this by dividing abstract states; growing the tree. In order to grow the tree, we need to know which leaves should be divided. A leaf should be divided when the utility of performing an abstract action is non-constant across the leaf.

We can use the trajectories sampled earlier to get point estimates of the $T$ function defined in equation 6. We assume that the abstract value function is a good approximation of the real value function, $V$. The sampled trajectories allow us to extimate that part of the $T$ function within the current abstract state:

$$\hat{T}_{\mathcal{S}}(s_i, \mathbf{a_j}) = r + \gamma^t V(\mathbf{s}_{stop})$$

If a statistical difference across an abstract state is detected in the $\hat{T}(s, \mathbf{a})$ estimates for any $\mathbf{a}$, or in $\text{argmax}_a \hat{T}(s, a)$, then that abstract state is divided in two. We also test to see if the action with the highest $\hat{T}$ In [3] we used a non-parametric test, the Kolmogorov-Smirnov test. The results in this paper are with a Minimum Description Length based test, described in [12]. The division that maximises the statistical difference between the two sides is chosen.

Once a division has been introduced, all trajectories sampled from the leaf that was split are discarded, a new set of trajectories is sampled in each of the new leaves, and the

**Table 1.** Procedure $\mathrm{TTree}(\mathcal{S}, \mathbf{A}, G, \gamma)$

```
1: tree ← a new tree with a single leaf corresponding to S
2: loop
3:    S_a ← {s_1, ..., s_{N_a}} sampled from S
4:    for all s ∈ S_a do
5:       SampleTrajectory(s, tree, A, G, γ) {see Table 2}
6:    end for
7:    UpdateAbstractSMDP(tree, A, G, γ) {see Table 3}
8:    D^T ← ∅ {Reset split data set. D^T is a set of states with associated T̂ estimates.}
9:    D^a ← ∅
10:   for all leaves l and associated points p do {a point contains a set of trajectories starting in
      the same state}
11:      T̂(s_start, .) ← ∅ {T̂(s_start, .) is a new array of size |A|}
12:      for all trajectories in p, ⟨s_start, a, s_stop, t, r⟩ do {N_t trajectories for each of |A| actions}
13:         l_stop ← LeafContaining(tree, s_stop)
14:         T̂(s_start, a) ← T̂(s_start, a) + (r + γ^t V(l_stop))/N_t
15:      end for
16:      D^T ← D^T ∪ {⟨s_start, T̂⟩} {add T̂ estimates to data set}
17:      k ← argmax_a T̂(s_start, a)
18:      D^a ← D^a ∪ {⟨s, k⟩} {add best action to data set}
19:   end for
20:   for all new splits in the tree do
21:      EvaluateSplit(D^T ∪ D^a) {Use the splitting criterion to evaluate this split to see if either
         T̂, for any action, or k varies across a leaf}
22:   end for
23:   if ShouldSplit(D^T ∪ D^a) then {Evaluate the best split using the stopping criterion}
24:      Introduce best split into tree
25:      Throw out all sample points, p, in the leaf that was split
26:   end if
27: end loop
```

algorithm iterates. The complete algorithm is shown as Procedure $\mathrm{TTree}(\mathcal{S}, \mathbf{A}, G, \gamma)$ in Table 1. The various constants referred to are defined in Table 4.

## 4 Empirical Results

We have evaluated TTree in the Towers of Hanoi domain. This domain is well known in the classical planning literature for the hierarchical structure of the solution; temporal abstraction should work well. We compared TTree against two other algorithms, a well known algorithm without abstraction, the Prioritized Sweeping algorithm [13] and an algorithm similar to TTree that performs only state abstraction, the Continuous U Tree algorithm [3].

Figure 1 shows a comparison of Prioritized Sweeping and TTree in the Towers of Hanoi domain. The data shown are the averages over 15 trials. The error bars show one standard deviation. For each trial the expected discounted reward was measured periodically by running 250 trajectories from random start points. This was recorded

**Table 2.** Procedure SampleTrajectory($s_{start}$, *tree*, $\mathbf{A}$, $G$, $\gamma$)

Initialize new trajectory sample point, $p$, at $s_{start}$ {$p$ will store $N_t$ trajectories for each of the $|\mathbf{A}|$ actions}
$l \leftarrow \text{LeafContaining}(tree, s_{start})$
**for all** abstract actions $\mathbf{a} \in \mathbf{A}$ **do**
  let $\pi_{\mathbf{a}}$ be the base policy associated with $\mathbf{a}$
  **for** $j = 1$ to $N_t$ **do**
    $s \leftarrow s_{start}$
    $t_{total} \leftarrow 0$, $r_{total} \leftarrow 0$
    **repeat**
      $\langle s, t, r \rangle \leftarrow G(s, \pi_{\mathbf{a}}(s))$
      $t_{total} \leftarrow t_{total} + t$
      $r_{total} \leftarrow r_{total} + \gamma^{t_{total}} r$
    **until** $s \notin$, **or** $t_{total} >$ MAXTIME, **or** $\langle s', *, * \rangle = G(s, \pi_{\mathbf{a}}(s))$ is deterministic and $s = s'$, **or** $s$ is an absorbing state
    **if** the trajectory stopped because of a deterministic self transition **then**
      $r_{total} \leftarrow r_{total} + \gamma^{(t_{total}+t)} r/(1-\gamma^t)$
      $t_{total} \leftarrow \infty$
    **else if** the trajectory stopped because the final state was absorbing **then**
      $t_{total} \leftarrow \infty$
    **end if**
    $s_{stop} \leftarrow s$
    Add $\langle s_{start}, \mathbf{a}, s_{stop}, t_{total}, r_{total} \rangle$ to the trajectory list in $p$
  **end for**
**end for**
Add $p$ to $l$

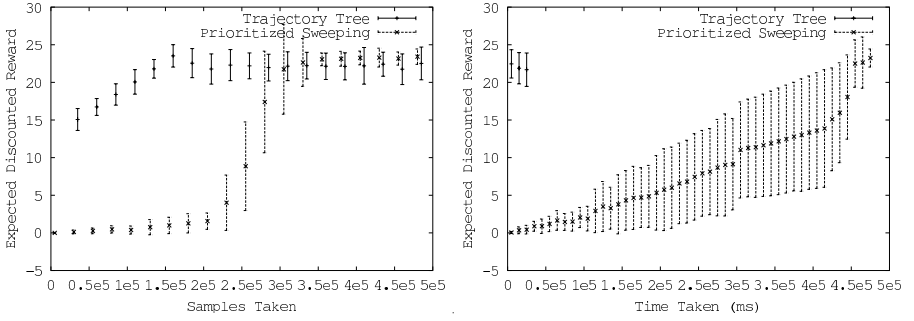
**Table 3.** Procedure UpdateAbstractSMDP(*tree*, $\mathbf{A}$, $G$, $\gamma$)

**for all** leaves $l$ with fewer than $N_l$ sample points **do**
  $\mathcal{S}_a \leftarrow \{s_1, \ldots, s_{N_a}\}$ sampled from $l$
  **for all** $s \in \mathcal{S}_a$ **do**
    SampleTrajectory($s$, *tree*, $\mathbf{A}$, $G$, $\gamma$) {see Table 2}
  **end for**
**end for**
$\mathcal{P} \leftarrow \emptyset$ {Reset abstract transition count}
**for all** leaves $l$ and associated points $p$ **do**
  **for all** trajectories, $\langle s_{start}, \mathbf{a}, s_{stop}, t_{total}, r_{total} \rangle$, in $p$ **do**
    $l_{stop} \leftarrow \text{LeafContaining}(tree, s_{stop})$
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{\langle l, \mathbf{a}, l_{stop}, t_{total}, r_{total} \rangle\}$
  **end for**
**end for**
Transform $\mathcal{P}$ into transition probabilities
Solve the abstract SMDP

along with the number of samples the algorithm had taken from the generative model and a time stamp. The y-axis in Figure 1 is the average expected discounted reward. The x-axis of (a) is the number of samples taken from the generative model. The x-axis of (b)

**Table 4.** Constants in the TTree algorithm

| Constant | Definition |
|---|---|
| $N_a$ | The number of trajectory start points sampled from the entire space each iteration |
| $N_l$ | The minimum number of trajectory start points sampled in each leaf |
| $N_t$ | The number of trajectories sampled per start point |
| MAXTIME | The number of time steps before a trajectory value is assumed to have converged. Usually chosen to keep $\gamma^{\text{MAXTIME}} r/(1-\gamma^t) < \epsilon$, where $r$ and $t$ are the largest reward and smallest time step, and $\epsilon$ is an acceptable error |




(a) Reward vs. Samples (b) Reward vs. Time

**Fig. 1.** (a) A plot of Expected Reward vs. Number of Sample transitions taken from the world. (b) The same data plotted against time instead of the number of samples

is the time stamp. In (b) the TTree data finishes significantly earlier than the Prioritized Sweeping data; TTree takes significantly less time per sample. Continuous U Tree results are not shown as that algorithm was unable to solve the problem.

The Towers of Hanoi domain had 8 discs. We had a discount factor, $\gamma = 0.99$. TTree was given policies for the three 7 disc problems. The TTree constants were, $N_a = 20, N_l = 20, N_t = 1$ and MAXSTEPS $= 400$. Prioritized Sweeping used Boltzmann exploration with carefully tuned parameters ($\gamma$ was also tuned to help Prioritized Sweeping). The tuning of the parameters for Prioritized Sweeping took significantly longer than for TTree.

We also tested Continuous U Tree and TTree on smaller problems without additional macros. TTree with only the automatically generated abstract actions was able to solve more problems than Continuous U Tree. We attribute this to the fact that the Towers of Hanoi is particularly bad for U Tree style state abstraction. In U Tree the same action is always chosen in a leaf. However, it is never legal to perform the same action twice in a row in Towers of Hanoi. TTree is able to solve these problems because the automatically generated random abstract action allows it to gather more useful data than Continuous U Tree.

## 5 Conclusion

We have described the TTree algorithm for combining state and temporal abstraction in Semi-Markov Decision Problems. We have given some detailed examples of the execution of the algorithm on two separate tasks. We have also supplied empirical results that show the algorithm is more effective in practice than another state abstraction algorithm, and that when extra macros are supplied, TTree is able to make use of these to further improve its results.

## References

1. Puterman, M.L.: Markov Decision Processes : Discrete stochastic dynamic programming. John Wiley & Sons (1994)
2. Chapman, D., Kaelbling, L.P.: Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In: Proceedings of IJCAI-91. (1991)
3. Uther, W.T.B., Veloso, M.M.: Tree based discretization for continuous state space reinforcement learning. In: Proceedings of AAAI-98. (1998)
4. Munos, R., Moore, A.W.: Variable resolution discretization for high-accuracy solutions of optimal control problems. In: Proceedings of IJCAI-99. (1999)
5. Sutton, R.S., Precup, D., Singh, S.: Intra-option learning about temporally abstract actions. In: Proceedings of ICML98, Morgan Kaufmann (1998) 556–564
6. Dietterich, T.G.: The MAXQ method for hierarchical reinforcement learning. In: Proceedings of ICML98, Morgan Kaufmann (1998)
7. Parr, R.S., Russell, S.: Reinforcement learning with hierarchies of machines. In: Neural and Information Processing Systems (NIPS-98). Volume 10., MIT Press (1998)
8. Baird, L.C.: Residual algorithms: Reinforcement learning with function approximation. In Prieditis, A., Russell, S., eds.: Proceedings of ICML95, Morgan Kaufmann (1995)
9. Ng, A.Y., Jordan, M.: PEGASUS: A policy search method for large MDPs and POMDPs. In: Proceedings of UAI00. (2000)
10. Hengst, B.: Generating hierarchical structure in reinforcement learning from state variables. In: Proceedings of PRICAI 2000. Volume 1886 of Lecture Notes in Computer Science., Springer (2000)
11. McGovern, A., Barto, A.G.: Automatic discovery of subgoals in reinforcement learning using diverse density. In: Proceedings of ICML01. (2001)
12. Uther, W.T.B., Veloso, M.M.: The lumberjack algorithm for learning linked decision forests. In: Proceedings of PRICAI 2000. Volume 1886 of Lecture Notes in Computer Science., Springer (2000)
13. Moore, A., Atkeson, C.G.: Prioritized sweeping: Reinforcement learning with less data and less real time. Machine Learning **13** (1993)

# Ontology-Driven Induction of Decision Trees at Multiple Levels of Abstraction

Jun Zhang, Adrian Silvescu, and Vasant Honavar

Artificial Intelligence Research Laboratory
Department of Computer Science
Iowa State University
Ames, Iowa 50011-1040
USA
{jzhang, silvescu, honavar}@cs.iastate.edu
http://www.cs.iastate.edu/~honavar/aigroup.html

**Abstract.** Most learning algorithms for data-driven induction of pattern classifiers (e.g., the decision tree algorithm), typically represent input patterns at a single level of abstraction – usually in the form of an ordered tuple of attribute values. However, in many applications of inductive learning – e.g., scientific discovery, users often need to explore a data set at multiple levels of abstraction, and from different points of view. Each point of view corresponds to a set of ontological (and representational) commitments regarding the domain of interest. The choice of an ontology induces a set of representatios of the data and a set of transformations of the hypothesis space. This paper formalizes the problem of inductive learning using ontologies and data; describes an ontology-driven decision tree learning algorithm to learn classification rules at multiple levels of abstraction; and presents preliminary results to demonstrate the feasibility of the proposed approach.

## 1 Introduction

Inductive learning algorithms (e.g., decision tree learning) offer a powerful approach to data-driven discovery of complex, a-priori unknown relationships (e.g., classifiers) from data. Most learning algorithms for data-driven induction of pattern classifiers (e.g., the decision tree algorithm), typically represent input patterns at a single level of abstraction – usually in the form of an ordered tuple of attribute values. They typically assume that each pattern belongs to one of a set of disjoint classes. Thus, any relationships might exist between the different values of an attribute or relationships between classes (e.g., a hierarchically nested class structure) are ignored. In contrast, data-driven knowledge discovery in practice, occurs within a *context*, or under certain *ontological commitments* on the part of the learner. The learner's ontology (i.e., assumptions concerning *things* that exist in the *world*) determines the choice of *terms* and *relationships* among terms (or more generally, *concepts*) that are used to describe the domain of interest and their intended correspondence with objects and properties of the world [11]. This is particularly true in scientific

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 316–323, 2002.

applications of machine learning where specific ontological and representational commitments often reflect prior knowledge and working assumptions of scientists. When several independently generated and managed data repositories are to be used as sources of data in a learning task, the ontological commitments that are implicit in the design of the data repositories may or may not correspond to those of the user (typically a scientist familiar with the domain e.g., a biologist) in a given context [6], [7]. For example, in one context, the scientist may not consider it necessary to distinguish between different subfamilies of a family of proteins or different types of sequence patterns or structural features of proteins. In other cases, such distinctions may be desirable. In computational characterization of protein sequence-structure-function relationships, it is often useful to consider alternative representations of protein sequences and different notions of protein function [2]. In scientific discovery applications, because users often need to examine data in *different contexts from different perspectives* and *at different levels of abstraction*, there is no single universal ontology that can serve all users, or for that matter, even a single user, in every context. Hence, methods for learning from ontologies and data are needed to support knowledge acquisition from heterogeneous distributed data.

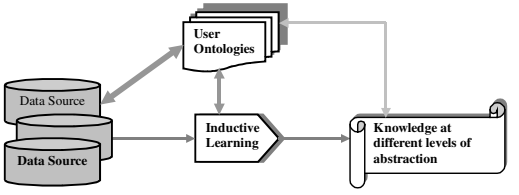



**Fig. 1.** Ontology-driven inductive learning

Making ontological commitments (that are typically implicit in a data set) explicit enables users to explore data from multiple perspectives, and at different levels of abstraction. Some aspects of ontology guided learning have received attention in the literature. Walker [13] first used the concept taxonomies in information retrieval from large database. Han et al. [4] proposed attribute-oriented induction of multi-level classification rules using background knowledge in the form of concept hierarchies. They also proposed a method to discover association rules at multiple levels of abstraction. Quinlan [9] suggested pre-processing approaches to deal with tree-structured attributes (ISA hierarchy), re-encoding the training examples in terms of an equivalent set of purely nominal attributes. Almuallim [1] proposed handling tree-structured attributes directly by routing examples in hierarchies, which count the class frequency of every concept node, then apply decision tree learning algorithm to score and find the best concept node in the hierarchy to build the decision tree. Taylor et al [12] proposed an algorithm for rule learning using taxonomies and data.

Against this background, it is of interest to formalize the problem of learning from ontologies and data and to explore the design space of algorithms for data-driven knowledge acquisition using *explicitly specified* ontologies. In this

paper, we formalize the problem of inductive learning using ontologies (as a form of background knowledge or working assumptions) and data. We present an ontology-driven decision tree learning algorithm to learn classification rules at multiple levels of abstraction. We present some preliminary results to demonstrate the feasibility of the proposed approach. We briefly examine several variations of the proposed approach and a general strategy for transforming traditional inductive learning algorithms into ontology-guided inductive learning algorithms for data-driven discovery of relationships at multiple levels of abstraction.

## 2 Role of Ontologies in Learning from Data

An ontology specifies the terms or concepts and relationships among terms and their intended correspondence to objects and entities that exist in the world [3], [11]. A formal ontology is specified by a collection of names for concept and relation types organized in a partial ordering by the type-subtype relation [11]. In philosophy, an ontology corresponds to a nomenclature of all things (entities, properties of entities, and relations among entities) that exist and as such, there is no room for multiple ontologies. However, such a view is untenable in practice. Consequently, we adopt the position that an ontology corresponds to a particular conceptualization of the world from a specific point of view.

Syntactically, given a logical language $L$, an ontology is a tuple $< V, A >$, where the vocabulary $V \subset S_p$ is some subset of the predicate symbols of$L$ and the axioms $A \subset W$ are a subset of the well-formed formulas of $L$ [5]. Taxonomies that specify hierarchical relationships among concepts in a domain of interest are among some of the most commonly used ontologies. Normally, we would draw this mapping in the form of a tree, or a directed acyclic graph (DAG).

As is usually the case in formulations of inductive learning problems, we will assume that each instance is described by a tuple of attribute values and that a concept corresponds to a set of instances that satisfy specific constraints on the values of their attributes. Note that there is a certain duality between attributes and concepts. For instance, instances in a particular domain may be described in terms of two attributes: *color* and *size*. Now, *blue*, a possible value for the attribute *color* is itself a concept (composed out of all the instances that have color blue). Thus, it is possible for each attribute to have an ontology associated with it.

In what follows, we consider several cases (ordered by their complexity) in which ontologies may play a role in learning from data.

a) Each attribute has associated with it, an ontology in the form of a hierarchy. A hierarchical ontology is the simplest form of ontology, and is typically represented by a tree. Every node in the tree represents a concept. The topmost root concept is the name of the corresponding attribute. Every link in the tree represents an interrelationship between two nodes. The interrelationship between concepts could be the relation of ISA, Instance-Of, or Part-Of. For an attribute with a finite domain of possible values, each value corresponds

to a primitive concept (or a leaf node) in a hierarchical ontology. An attribute without a hierarchically structured ontology corresponds to a single level taxonomy with attribute name as root, and the attribute values as the children leaf nodes of root.

b) Each attribute has a single ontology associated with it. However, concepts that appear in ontologies associated with different attributes can be related. This results in ontologies that can be represented using directed acyclic graphs (DAGs). (this would yield more complicated ontologies than a set of competing hierachies).

The next section describes an ontology-driven decision tree construction algorithm for the first of the four cases outlined above.

## 3 Ontology-Guided Decision Tree Learning Algorithm

We consider decision tree learning in a scenario in which each attribute has a single hierarchically structured ontology (e.g., a concept taxonomy) associated with it and each instance is labeled with one of $m$ disjoint class labels. Ontology-driven Decision Tree (ODT) learning algorithm is a top-down multi-level ontology (concept hierarchy) guided search in a hypothesis space of decision trees.

Recall that the basic decision tree algorithm recursively selects at each step, an attribute from a set of candidate attributes based on an information gain criterion [9]. Thus, each node in a partially constructed decision tree has associated with it, a set of candidate attributes to choose from for growing the tree rooted at that node.

In our case, each attribute has associated with it, a hierarchically structured taxonomy over possible values of the attribute. Thus, the learning algorithm has to choose not just a particular attribute, but also an appropriate level of abstraction in the taxonomy. The basic idea behind the algorithm is to start with abstract attributes (i.e., groupings of attribute values that corresponds to nodes that appear at higher levels of a hierarchically structured attribute value taxonomy). Thus, each node of a partially constructed decision tree has associated with it, a set of candidate attributes drawn from the taxonomies associated with each of the individual attributes. The algorithm maintains for each node in the partially constructed decision tree, a set of pointers to nodes on the frontier, computes information gain for the corresponding attributes, and selects from the set of candidate attributes under consideration, one with the largest information gain.

It is useful to introduce some notation to help describe our algorithm. Let the set of attributes used to describe instances in the data set be $A = \{A_1, A_2, ..., A_n\}$. Let the set of class labels be $O = \{O_1, O_2, ..., O_m\}$. Each attribute Ai, has associated with it, a corresponding taxonomy $T_i$. The set of ontologies is denoted by $T = \{T_1, T_2, ..., T_n\}$. Note that the root node of taxonomy $T_i$ is $A_i$. Let $\Psi(c)$ denote the children of a node $c$. The training data set is denoted by $S$. Each leaf node of a partially constructed decision tree has associated with it, a subset of the training data set. We will associate with each such set of examples,

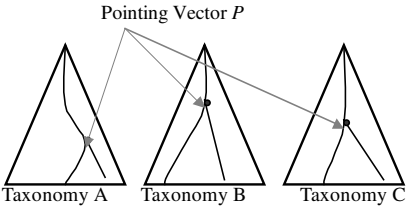



**Fig. 2.** Vector of Pointers: concept frontier across different attribute taxonomies

$n$ pointers that point to $n$ concepts in $n$ taxonomies (one for each attribute). Let $P = \{p_1, p_2, ..., p_n\}$ denote such a vector of pointers where $p_i$ is concept in taxonomy $T_i$. A vector of pointers is called an *ending vector* if each $p_i$ is a leaf node in the corresponding taxonomy $T_i$. We use $\Phi(P) = true$ to denote that $P$ is an ending vector. Note that $\Phi(P) = true$ if and only if $\forall p_i \in P$, $\Psi(p_i) = \{\}$.

Our current implementation selects an attribute from a set of candidate attributes (specified by the vector of pointers) that yields the maximum reduction in entropy as the splitting criterion to partition the dataset [9]. However, other splitting criteria could also be adopted directly in our algorithm (e.g. Gini Index, one-sided purity, one-sided extremes, etc).

**ODT algorithm:**
*ODT (Examples $S$, Attributes $A$, Taxonomies $T$, Class labels $O$, Vector of Pointers $P$, Default $Df$)*

1. If decision tree is $NULL$ Then create a root node for decision tree, set all examples to $S$, and set $P = \{A_1, A_2, ..., A_n\}$, and set $Df = Majority\_Class(S)$
2. If $S$ is empty Then assign the label $Df$ to the node.
   Else If every instance in $S$ has the same class label $o$ Then Return($o$)
   Else If $\Phi(P)$ is true Then assign the label $Df$ to the node.
3. Calculate the best attribute $B_j$ and best concept $b$ by calling function *Choose-Best(P, S, A, O)*
4. Set the best partition value set $Bvalue = \Psi(b)$
5. Partition the examples S using the concepts in $Bvalue$
   For each value $V_i$ in Bvalue Do
   $S_i$ = subset of S with concept $V_i$
   $j$ = order of $B_j$ in $A$
   update the Pointing Vector $P$ to $P'$ by substituting $p_j$ for $V_i$
   construct the subtree by calling$ODT(S_i, A, O, P', Majority_Value(S_i))$
   add new branch with label $V_i$ and connect to its subtree.
   End
6. Return the Decision Tree

*Choose-Best(PointingVectors $P$, Examples $S$, Attributes $A$, Class labels $O$)*
/* Returns the attribute whose expansion will yield the best information gain, selected from P = $\{p_1, ...p_n\}$ */
Return $\underset{i}{\text{argmax}}\ Gain(S, p_i)$

The ontology-driven decision tree algorithm can be viewed as a best-first search through the hypothesis space of decision trees defined with respect to a set of attribute taxonomies.

### 3.1 Illustration of the Working of the ODT Algorithm

The preliminary test of our algorithm is based on a simple customer purchase database that used in Taylor's paper [12]. Each of the attributes used to describe instances in this data set has a taxonomy associated with it. The two taxonomies are ISA hierarchies for Beverage and Snack. For concepts in the Beverage taxonomy, there are three different levels of abstraction, and in the Snack taxonomy, we have two different levels of abstraction. The class has three values: Young, Middle-aged, and Old. Figure 3 shows the two taxonomies, and Figure 4 shows the dataset and the induced tree.

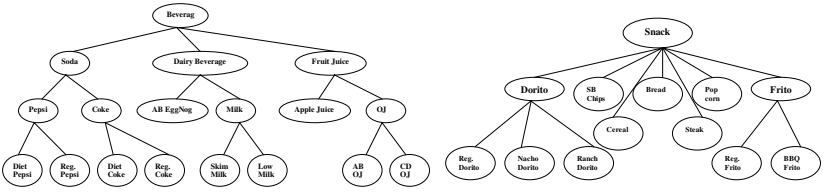



**Fig. 3.** Two taxonomies defined over two attributes

| Customer | Item1 | Item 2 | Class |
|---|---|---|---|
| 1 | Diet Coke | Ranch Dorito | Young |
| 2 | AB OJ | CD Cereal | Old |
| 3 | Reg Coke | Reg Dorito | Young |
| 4 | Reg Coke | SB Chips | Mid-Aged |
| 5 | Diet Coke | Nacho Dorito | Young |
| 6 | Diet Pepsi | BBQ Frito | Mid-Aged |
| 7 | Reg Pepsi | Reg Frito | Mid-Aged |
| 8 | Skim Milk | CD Cereal | Old |
| 9 | Reg Pepsi | BBQ Frito | Mid-Aged |
| 10 | CD OJ | Bread | Old |
| 11 | Reg Pepsi | Popcorn | Young |
| 12 | AB Egg Nog | CD Steak | Old |

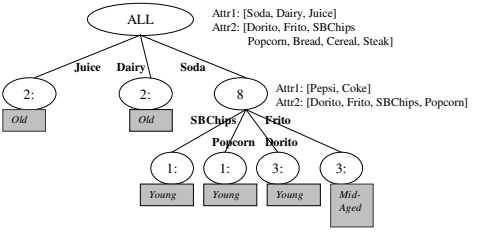



**Fig. 4.** Sample customer purchase database and the induced decision tree

We start our search with the Pointing Vectors pointing to the root of both taxonomies. The information gain associated with the attribute Beverage is higher than that associated with Snack. Consequently, the attribute corresponding to the root of the Beverage hierarchy is selected to partition the original data. This yields a 3-way split at the root of the decision tree. Two examples are classified as Old on the basis of the attribute corresponding to the concept "Dairy Beverage", and two examples are classified as Old on the basis of the attributed corresponding to the concept concept "Fruit Juice". The remaining eight examples need to be partitioned further. The first element $p_1$ in Pointing Vector for these eight examples changes to "Soda" in the Beverage Taxonomy, and hence the possible choice of attribute values will be [Pepsi, Coke]. While the second element $p_2$ continues to the root of the Snack Taxonomy, the possible attribute value set will include all the available concepts in the first level of this taxonomy.

This time $p_2$ yields a better value for information gain and all eight examples are correctly classified. The resulting decision tree corresponds to the rules: *If* Soda and Dorito *Then* Young; *If* Soda and Frito *Then* Midle-aged; *If* Dairy *Then* Old; *If* Juice *Then* Old. Note that many of the rules discriminate among instances belonging to the three classes using attributes that correspond to concepts that reside at higher levels of the corresponding attribute taxonomies.

## 4 Summary and Discussion

Many practical applications of machine learning (e.g., data-driven scientific discovery in computational biology [2] call for exploration of a data set from multiple perspectives (that correspond to multiple ontologies). Different ontologies induce different representations of the data and transformations of the hypothesis space. For example, hierarchically structured taxonomies over values of attributes facilitate discovery of classifiers at different levels of abstraction. The work described in this paper represents a tentative first step toward formulating the problem of ontology-guided data-driven knowledge discovery. We have demonstrated an extension of the standard decision tree learning algorithm that can exploit user-supplied ontologies to induce classification rules at higher levels of abstraction.

Work in progress is aimed at:

a) Systematic experimental evaluation of the proposed algorithm on real-world data sets that are encountered in computational biology (e.g., data-driven characterization of macromolecular sequence-structure-function relationships), text classification, among others.
b) Extensions of the proposed approach to accommodate use of multiple hierarchically structured ontologies for each attribute, as well as DAG-structured (as opposed to tree-structured) ontologies.

In related work, we are exploring ontology-guided learning algorithms for domains in which:

a) Each class (target attribute) has a tree-structured concept hierarchy (e.g., a taxonomy) associated with it. For example, in a pattern classification task, it may be necessary to classify an instance at different levels of abstraction (e.g., a soft drink into Pepsi, Cola, carbonated beverage). Higher level concepts correspond to generalizations of the basic level target concepts.
b) Each instance may belong to more than one class. (For instance, an individual may be classified as a parent, student, wife, friend. A protein may have multiple not necessarily mutually exclusive functions) [10].
c) There is a need to integrate information from multiple heterogeneous, autonomous, distributed data and knowledge sources from different ontological view points [6],[7] (e.g., in scientific discovery environments).

**Acknowledgements.** This research was supported in part by grants from the National Science Foundation (9982341, 0087152, 9972653) and Pioneer Hi-Bred to Vasant Honavar and the Iowa State University (ISU) Graduate College. This research has benefited from discussions with members of the ISU Artificial Intelligence Research Laboratory.

## References


1. Almuallim H., Akiba, Y., Kaneda, S.: On Handling Tree-Structured Attributes. Proceedings of the Twelfth International Conference on Machine Learning (1995)
2. Andorf, C., Dobbs, D., Honavar, V.: Discovering Protein Function Classification Rules from Reduced Alphabet Representations of Protein Sequences. Proceedings of the Conference on Computational Biology and Genome Informatics. Durham, North Carolina (2002)
3. Gruber T. R.: A translation approach to portable ontology specifications. Knowledge Acquisition, 5(2):199-220, (1993)
4. Han, J., Fu, Y.: Exploration of the Power of Attribute-Oriented Induction in Data Mining. U.M. Fayyad, el al. (eds.), Advances in Knowledge Discovery and Data Mining. AAAI/MIT Press (1996)
5. Heflin, J., Hendler, J., Luke, S.: Coping with Changing Ontologies in a Distributed Environment. Ontology Management. Papers from the AAAI Workshop. WS-99-13. AAAI Press (1999)
6. Honavar, V., Silvescu, A., Reinoso-Castillo, J., Andorf, C., Dobbs, D.: Ontology-Driven Information Extraction and Knowledge Acquisition from Heterogeneous, Distributed Biological Data Sources. Proceedings of the IJCAI-2001 Workshop on Knowledge Discovery from Heterogeneous, Distributed, Autonomous, Dynamic Data and Knowledge Sources. (2001)
7. Reinoso-Castillo, J.: An Ontology-Driven Query-Centric Approach to Information Integration from Heterogeneous, Distributed, Autonomous Data Sources. Masters Thesis. Artificial Intelligence Research Laboratory, Iowa State University, June 2002.
8. Kamber, M., Winstone, L., Gong, W., Cheng, S., Han, J.: Generalization and Decision Tree Induction: Efficient Classification in Data Mining, Proc. of 1997 Int'l Workshop on Research Issues on Data Engineering (RIDE'97) Birmingham, England, April (1997)
9. Quinlan, J. R.: C4.5: Programs for machine learning. San Mateo, CA: Morgan Kaufmann (1992)
10. Silvescu, A., Caragea, D., and Honavar, V.: Learning Decision Tree Classifiers When Classes are not Mutually Exclusive. To appear. (2002)
11. Sowa, J.: Knowledge Representation: Logical, Philosophical, and Computational Foundations. Brooks Cole Publishing Co., Pacific Grove, CA (2000)
12. Taylor, M., Stoffel, K., Hendler, J.: Ontology-based Induction of High Level Classification Rules. SIGMOD Data Mining and Knowledge Discovery workshop proceedings. Tuscon, Arizona (1997)
13. Walker, A.: On retrieval from a small version of a large database. VLDB Conference (1980)

# Abstracting Imperfect Information Game Trees

Darse Billings

University of Alberta Department of Computing Science
Edmonton, Alberta, Canada
darse@cs.ualberta.ca http://www.cs.ualberta.ca/~darse

Modern game programs have achieved spectacular success in many games, such as checkers, Othello, and chess. In contrast, games with hidden information, like poker and bridge, have a fundamentally different structure, and game-playing programs have not had as much success to date.

Unlike perfect information games, the nodes of an imperfect information game tree are not independent, being grouped into *information sets* that are indistinguishable from the perspective of one player. Furthermore, it is not sufficient to always choose the same option in a given situation – the best play may dictate a *mixed strategy*, varying randomly between certain choices to avoid revealing information to the opponent.

Since the nodes in different subtrees are not independent, we cannot use a divide and conquer algorithm – we must solve the tree as a whole.

The conventional method used in game theory is to convert the game tree into a large system of linear equations, and then solve that problem with linear programming (LP). This produces an optimal strategy for each player to obtain the Nash equilibrium value for the game.

The maximum size of solvable problems is still quite modest, on the order of a few million nodes. In contrast, even greatly simplified variations of poker have game trees with billions or trillions of nodes. The game of Texas Hold'em (the poker variant used to determine the World Champion) has over $10^{18}$ nodes in the 2-player game tree, which is obviously far beyond the range of feasible computation.

However, the daunting size of these irreducible game trees does not preclude the possibility of finding good approximations, resulting in a strong betting strategy and a high level of play in practice. To do this, we define abstracted game trees which retain all the key properties of the underlying real game tree. Solutions to moderate sized games may provide near-optimal strategies for the real game, with a bounded degree of error. Failing that, they can at least suggest pseudo-optimal strategies which are superior to the current rule-based betting schemes.

There are numerous ways to do abstraction on poker game trees, each having its own advantages and disadvantages. These trade-offs need to be explored to find the most effective balance given the limitations of computation time and memory. For Hold'em, we have investigated several approaches, the most important ones being *coarser granularity* by hand classification, and *truncated games*.

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 324–325, 2002.

To change the granularity, we combine a group of nodes into a single class, treating it as a single node in the abstracted game. This is most naturally done along the lines of the information sets, combining hands into groups that will all be played in the same way. This requires a mapping function to designate which hands go into which classes, and some error will result if fundamentally distinct hands are merged into the same class. The mapping function is used to translate the real game onto a fully abstracted game, which has a moderate number of "macro-nodes". This game is then solved exactly, and the resulting strategies are applicable to the real game after being re-translated onto actual hands by the same mapping function.

The collection of specific hands into bins can be taken to extremes, with just a few bins to distinguish a very small number of hand types. This will produce an abstracted game that is small enough to compute directly, but the consequences of the very crude abstraction may be severe. Increasing the number of hand types will expand the size of the problem, which might then require other abstraction techniques to compensate, scaling it back down to the scope of feasibility.

There are several ways to create truncated games, again with interesting trade-offs between game tree size and effectiveness. Without getting into too many details, we have experimented with all of the following: first-round nullification (fixed player actions and chance node outcome); reduced betting structure within each round (maximum number of bets); one-round models with expected-value leaf nodes; two-round models with expected-value leaf nodes; combining two or more future betting rounds into one; and full betting structure models with exact-value leaf nodes.

There are many aspects of abstraction on imperfect information game trees still to be explored, but the preliminary results are very encouraging.

## References

[1] D. Billings. Computer poker. Master's thesis, University of Alberta, 1995.

[2] D. Billings, D. Papp, J. Schaeffer, and D. Szafron. Opponent modeling in poker. In *AAAI National Conference*, pp 493–499, 1998.

[3] D. Billings, L. Pena, J. Schaeffer, and D. Szafron. Using probabilistic knowledge and simulation to play poker. In *AAAI National Conference*, pp 697–703, 1999.

[4] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron. The challenge of poker. In *Artificial Intelligence*, 134(1-2), pp 201–240, 2002.

[5] D. Koller and A. Pfeffer. Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94(1), pp 167–215, 1997.

# Using Abstraction for Heuristic Search and Planning

## Research Summary

Adi Botea

Department of Computing Science
University of Alberta
Edmonton, Canada T6G 2E8
adib@cs.ualberta.ca

My research interests are in the areas of planning, computer games, and heuristic search. My objectives include using abstraction to solve hard heuristic search problems. Classical heuristic search has been successful for games like Chess and Checkers, but seems to be of limited value in games such as Go and Shogi, and puzzles such as Sokoban. My work is focused on developing a planning approach for Sokoban, aiming to overcome the limitations exhibited by current approaches. In Sokoban, a man in a maze has to push stones from their current location to designated goal locations. The game is difficult for a computer for several reasons including deadlocks (positions from which no goal state can be reached), large branching factor (can be over 100), long optimal solutions (can be over 600 moves), and an expensive lower-bound heuristic estimator, which limits search speed.

We have introduced *Abstract Sokoban*, a highly abstracted representation of the game that replaces the huge initial search space by several smaller search spaces [1]. Since humans still perform much better than computers in Sokoban, our approach uses concepts that humans also consider when playing the game. For instance, our abstract Sokoban representation uses a preprocessing phase to decompose the puzzle into two types of objects: *rooms* and *tunnels*. The initial problem is transformed into several simpler problems in a divide-and-conquer manner, as illustrated in Fig. 1. In the global problem, which we address by planning, the abstraction is obtained by mapping the maze to a small graph of rooms connected by tunnels. At the local level, rooms and tunnels are further abstracted, defining *abstract states* that model their internal stone configurations. One abstract state represents all the *equivalent* configurations that can be obtained from each other in such a way that no stone leaves or enters the room or tunnel.

By splitting the problem into a local component (moves within a room) and a global component (moves between rooms and tunnels), the initial search space is transformed into a hierarchy of smaller spaces, each with a corresponding reduction in the search effort required. The experimental results show that the abstraction has the potential for an exponential reduction in the size of the search space explored.

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 326–327, 2002.

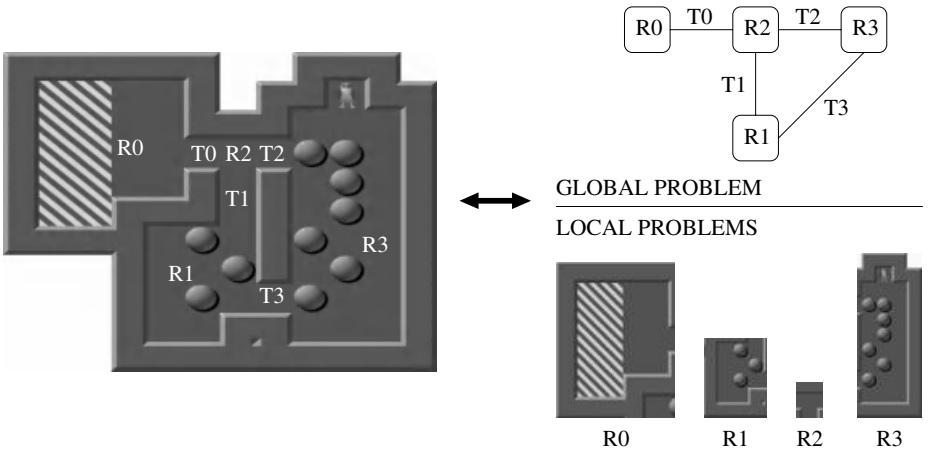



**Fig. 1.** A problem is decomposed into several abstract sub-problems. There is one global problem as well as one local problem for each room. Rooms and tunnels are denoted by $R$ and $T$, respectively.

There are many research directions that we plan to explore with abstract Sokoban. An enhancement that we expect to have a great impact on the system performance is a smarter decomposition of the maze into rooms and tunnels. While our current heuristic rule that guides this process is quite rigid, it can be replaced by a more general strategy aiming to optimize several parameters. To reduce the global search space, the number of rooms and tunnels as well as their interactions should be minimized. On the other hand, rooms should be small enough to solve the corresponding local problems. The global planning search space can be further simplified by detecting large deadlocks that involve interactions between several rooms and tunnels. Another important future research topic is to try our ideas in real-life planning domains (e.g., robotics related). Automatic abstraction of planning domains is also an extension of our work that we plan to work on in the future.

## References

[1] A. Botea, M. Müller, and J. Schaeffer. Using Abstraction for Planning in Sokoban. 2002. Submitted to CG 2002. `http://www.cs.ualberta.ca/~adib/`.

# Approximation Techniques in Multiagent Learning

Michael Bowling

Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213-3891

Research in multiagent systems includes the investigation of algorithms that select actions for multiple agents coexisting in the same environment. Multiagent systems are becoming increasingly relevant within artificial intelligence, as software and robotic agents become more prevalent. Robotic soccer, disaster mitigation and rescue, automated driving, and information and e-commerce agents are examples of challenging multiagent domains. As the automation trend continues, we need robust algorithms for coordinating multiple agents, and for effectively responding to other external agents.

Multiagent domains require determining a course of action for an agent just as in single-agent domains. Learning techniques have shown, in single-agent settings, to be a powerful tool that can discover and exploit the dynamics of the environment and adapt to unforeseen difficulties in the task. These same benefits are needed in multiagent domains. Similarly, multiagent environments also face the problem of scaling. Many environments are simply too large for our limits on memory, computation, and training data. Approximation and abstraction techniques, which are an important topic in single-agent learning, are also relevant in settings with multiple agents.

My research explores the intersection of multiagent learning with issues of applying learning to large problems that require the use approximation, generalization, and abstraction techniques. This overlap raises many interesting research challenges as we try to scale multiagent learning to large problems. First is the problem of simultaneous learning. Traditional single-agent learning, both with and without approximation techniques, assume the world is Markovian. When more than one agent is learning in the same environment this property is violated as their action choices may change as they all adapt and learn. This also may mean that the optimal policy for an agent is changing as the other agent's action choices change. The game theoretic concept of equilibrium aids in understanding this simultaneous learning problem. This raises the second important challenge. If approximation techniques are necessary for learning to scale then "optimal" agents are not practical. This is one of the key assumptions that underlies equilibria, creating a tension between approximate solutions and the concept of equilibria. The third challenge is to actually combine multiagent learning algorithms and approximate algorithms, creating a learner capable of handling large multiagent environments. I will briefly mention some of my work examining these challenges.

*Rational and Convergent Learning.* One of the problems of simultaneous learning is that traditional techniques that are guaranteed to converge to an optimal solution may not converge at all in a multiagent domain. Other approaches try to learn equilibrium solutions directly (e.g., Minimax-Q [Littman, 1994]), and can be guaranteed to converge to an equilibrium. These approaches, though, are not rational, in the sense that they may converge to non-optimal solutions. For example, converging to the equilibrium solution

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 328–329, 2002.

in the children's game rock-paper-scissors is a very sub-optimal policy when the other player mostly plays rock. We have explored trying to gain the advantages of both rational and convergent learning. I have introduced the WoLF principle as a modification to rational algorithms to make them converge [Bowling and Veloso, 2002a]. There is both empirical results and theoretical results showing this is indeed a powerful technique. It also lends itself well to being combined with approximation solutions as I will describe below.

*Restricted Equilibria.* Approximation, abstraction, and generalization often sacrifice optimality for speed of learning. This loss in optimality has drastic consequences on the existence of equilibria when using these limiting techniques [Bowling and Veloso, 2002b]. We have examined these consequences showing that even very well-behaved limitations may destroy the existence of equilibria in multiagent learning problems. We have also identified some limited situations where equilibria continue to exist, e.g., team-games and single-controller stochastic games where the agents have certain requirements on their limitations.

*Approximation and Multiagent Learning.* Although equilibria may not exist when agents use techniques that prevent them from learning optimal solutions, these techniques are still necessary. I am currently examining the combination of the WoLF principle with a policy gradient ascent approach [Sutton et Al., 2000]. The results are very exciting [Bowling *et al.* 2002]. Not only could effective simultaneous learning be demonstrated on a very large problem, but the WoLF principle continued to have a powerful effect on convergence. The worst-case performance of learning over time was greatly reduced using WoLF, and the expected payoff while learning was far less erratic.

This research is still ongoing with a number of open challenges. Can WoLF and policy gradient techniques be applied to other large multiagent learning problems? Since equilibria don't exist in problems requiring sub-optimal techniques, what are other methods of evaluation? Can agents model the limitations of the other agents and exploit this knowledge? How can this learning and adaptation be built into real large-scale multiagent systems? I am continuing to explore the above challenges as well as these additional questions in my future research.

## Relevant Publications

[Bowling and Veloso, 1999] Michael Bowling and Manuela Veloso. Bounding the suboptimality of reusing subproblems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1340–1345, Stockholm, Sweden, August 1999. Morgan Kaufman.

[Bowling and Veloso, 2002a] Michael Bowling and Manuela Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 2002. In Press.

[Bowling and Veloso, 2002b] Michael Bowling and Manuela M. Veloso. Existence of multiagent equilibria with limited agents. Technical report CMU-CS-02-104, Computer Science Department, Carnegie Mellon University, 2002.

[Bowling *et al.* 2002] Michael Bowling, Rune Jensen, and Manuela Veloso. A formalization of equilibria for multiagent planning. In *AAAI Workshop on Planning with and for Multiagent Systems*, Edmonton, Canada, July 2002. To Appear.

# Abstraction and Reformulation in GraphPlan

Daniel Buettner

Constraint Systems Laboratory
Department of Computer Science and Engineering, 115 Ferguson Hall,
University of Nebraska-Lincoln, Lincoln NE 68588-0115
buettner@cse.unl.edu

My research is chiefly focused in the area of planning. Specifically, I am interested in extending the Graphplan [1] planning system. I intend to explore, design, and evaluate abstraction and reformulation techniques to expand the expressiveness of Graphplan and enhance the performance of the search process used for building a plan.

In particular, I intend to study the semantic and topology of the constraints involved in Graphplan extended to encompass sensory planning [2]. Then I will devise new mechanisms for generating dynamically compact families of robust solutions as described in [3,4,5].

## References

1. Blum, A., Furst, M.L.: Fast planning through planning graph analysis. Artificial Intelligence **90** (1997) 281–300
2. Weld, D.S., Anderson, C.R., Smith, D.E.: Extending graphplan to handle uncertainty and sensing actions. In: AAAI/IAAI. (1998) 897–904
3. Choueiry, B.Y., Noubir, G.: On the computation of local interchangeability in discrete constraint satisfaction problems. In: AAAI/IAAI. (1998) 326–333
4. Beckwith, A.M., Choueiry, B.Y.: On the Dynamic Detection of Interchangeability in Finite Constraint Satisfaction Problems. In Walsh, T., ed.: Proceedings of $7^{th}$ International Conference on Principle and Practice of Constraint Programming (CP'01). Volume 2239 of Lecture Notes in Computer Science., Paphos, Cyprus, Springer Verlag (2001) 760
5. Beckwith, A.M., Choueiry, B.Y., Zou, H.: How the Level of Interchangeability Embedded in a Finite Constraint Satisfaction Problem Affects the Performance of Search. In: AI 2001: Advances in Artificial Intelligence, $14th$ Australian Joint Conference on Artificial Intelligence. Lecture Notes in Artificial Intelligence LNAI 2256, Adelaide, Australia, Springer Verlag (2001) 50–61

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, p. 330, 2002.

# Abstract Reasoning for Planning and Coordination

Bradley J. Clement

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive, M/S 126-347
Pasadena, CA 91109-8099
bclement@aig.jpl.nasa.gov

## 1 Research Summary

As autonomous software and robotic systems (or *agents*) grow in complexity, they will increasingly need to communicate and coordinate with each other. These agents will need planned courses of action to achieve their goals while sharing limited resources. Our research addresses the problem of efficiently interleaving planning and coordination for multiple agents.

As part of our approach, we represent agents as having hierarchies of tasks that can be decomposed into executable primitive actions. Using task hierarchies, an agent can reason efficiently about its own goals and tasks (and those of others) at multiple levels of abstraction. By exploiting hierarchy, these agents can make planning and coordination decisions while avoiding complex computation involving unnecessary details of their tasks.

To reason at abstract levels, agents must be aware of the constraints an abstract task embodies in its potential decompositions. Thus, we provide algorithms that summarize these constraints (represented as propositional state conditions and metric resource usages) for each abstract task in an agent's library of hierarchical plans. This summary information can be derived offline for a domain of problems and used for any instance of tasks (or plans) assigned to the agents during coordination and planning. We also provide algorithms for reasoning about the concurrent interactions of abstract tasks, for identifying conflicts, and for resolving them.

We use these algorithms to build sound and complete refinement-based coordination and planning algorithms. We also integrate summary information with a local search planner/scheduler, showing how the benefits can be extended to different classes of planning algorithms. Complexity analyses and experiments show where abstract reasoning using summary information can reduce computation and communication exponentially along a number of dimensions for coordination, planning, and scheduling in finding a single agent's plan or in optimally coordinating the plans of multiple agents. In addition, we provide pruning techniques and heuristics for decomposition that can further dramatically reduce computation. This use of abstraction extends previous work by Korf and

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 331–332, 2002.

Knoblock, that assumes subproblems do not interact. Our algorithms do not make this assumption.

Overall, the developed techniques enable researchers and system designers to scale the capabilities of interleaved coordination, planning, and execution by providing agents with tools to reason efficiently about their plans at multiple levels of abstraction.

## References

1. B. Clement. *Abstract Reasoning for Multiagent Coordination and Planning.* PhD thesis, University of Michigan, Ann Arbor, 2002.
2. B. Clement, A. Barret, and R. Rabideau. Using abstraction in multi-rover scheduling. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics, and Automation in Space,* 2001.
3. B. Clement, A. Barrett, G. Rabideau, and E. Durfee. Using abstraction in planning and scheduling. In *Proceedings of the European Conference on Planning*, 2001.
4. B. Clement and E. Durfee. Theory for coordinating concurrent hierarchical planning agents. In *Proceedings of the National Conference on Artificial Intelligence*, pages 495–502, 1999.
5. B. Clement and E. Durfee. Top-down search for coordinating the hierarchical plans of multiple agents. In *Proceedings Internation Conference on Autonomous Agents*, 1999.
6. B. Clement and E. Durfee. Performance of coordinating concurrent hierarchical planning agents using summary information. In *Proceedings of the Workshop on Architectures, Theories, and Languages*, pages 213–227, 2000.

# Research Summary: Abstraction Techniques, and Their Value

Irit Askira Gelman

Department of Management Information Systems
Mclelland Hall 430
The University of Arizona
Askira@bpa.arizona.edu

A decision support system (DSS) is often shared by multiple decision-makers or analysts. Different users may vary in their viewpoints, and therefore, in particular, they may place different demands on a DSS model-base. Users may benefit if the system offers each a model that suits his/her special requirements.

My MSc thesis[1] [1] centers on the problem of automated modeling. This research is based on Simon's *causal ordering* theory [4], Iwasaki's extension of causal ordering to dynamic models [6],[7], the notion of *nearly decomposable systems* [5], and, related to it, Iwasaki's *equilibration* and *exogenization* time scale-based model abstraction techniques [6],[7]. The study offers a formal theoretical basis and algorithms for a group of model abstraction techniques. This way, for example, I develop a theoretical basis and algorithms for a generalization of Iwasaki's equilibration, and propose a new time scale-based model abstraction technique termed *quasi-exogenization* [1],[2]. In addition, the study analyzes the value of the proposed techniques in terms of reduced complexity, especially their influence on the number of causal cycles in the model. The results of this analysis vary.

My PhD research[2] [3] addresses a related problem This study compares the *informativeness* of an information set which is described in terms of the distributions of its predictions in different possible states, to the *informativeness* of another, essentially more precise, information set. The immediate motivation for this inquiry is the recent pervasiveness of integrated information systems, and information systems integration projects. The study aims to develop a broad, formal theory of the conditions in which more precise information can/cannot increase the accuracy and economic value of predictions, and identify general conditions in which more precise information can be particularly valuable. The inquiry is based on a model of information, often known as the *information structure* [8] model, and related theory in Information Economics [9] and Management Information Systems (MIS) [10].

[1] With Niv Ahituv of Tel Aviv University.
[2] With Dave Pingry of the University of Arizona.

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 333-334, 2002.

## References

1. Askira Gelman, I., Model Reduction: Extracting Simple and Adequate Mathematical Macro-econometric Model-Based Information. MSc Thesis, Department of Information Systems, Tel Aviv University, Tel Aviv, Israel (1998).
2. Askira Gelman, I., Model Abstractions that Address Time-Scale Differences among Decision-Makers." Accepted for publication by DSS Internet Age 2002 Conference, Cork, Ireland (2002).
3. Askira Gelman, I., A theory of the Economic Value of Information Systems Integration: An Information Economics Perspective. (2001)
4. Simon, H.A., On the Definition of the Causal Relation. Journal of Philosophy, 49 (1952) 517-528.
5. Simon, H.A. Ando, A., Aggregation of variables in Dynamic Systems. Econometrica, Vol. 29 (1961) 111-138.
6. Iwasaki, Y., Model Based Reasoning of Device behavior with Causal Ordering, PhD Thesis, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA (1988).
7. Iwasaki, Y., and Simon, H.A., Causality and Model Abstraction. Artificial Intelligence, No. 67 (1994) 143-194.
8. Marschak, J., Economics of Information Systems. Journal of American Statistical Association, Vol. 66, No. 333 (1971) 192-219.
9. McGuire, C.B., Comparisons of Information Structures. In: C.B., McGuire and R. Radner (eds.), Decision and Organization, University of Minnesota Press, 2nd edition (1986) 101-130.
10. Ahituv, N., Ronen, B., Orthogonal Information Structures - A Model to Evaluate the Information Provided by a Second Opinion. Decision Sciences, Vol. 19, No. 2 (1988) 255-268.

# Reformulation of Non-binary Constraints

Robert Glaubius

Constraint Systems Laboratory
Department of Computer Science and Engineering, 115 Ferguson Hall,
University of Nebraska-Lincoln, Lincoln NE 68588-0115
glaubius@cse.unl.edu

Constraint satisfaction problems (CSPs) frequently emerge in practical situations. Unfortunately these problems are likely to be computationally intractable. In practice, it is necessary to implement strategies that reduce the amount of computational effort expended in finding solutions to CSPs.

One area that provides a source of potential improvements is the problem formulation. The use of abstractions and reformulations plays an important role in simplifying problems. Our work focuses on the use of constraint reformulation and decomposition. In particular, we are currently experimenting with the use of non-binary constraints, and novel reformulations (such as constraint decomposition, as discussed in [1]) for reducing the arity of global constraints.

While a large corpus of work exists regarding binary constraints, the study of non-binary constraints has seen significantly less progress. While general strategies for transforming a non-binary CSP into a binary CSP exist [2], the resulting problem is often impractically large. This is especially true of real-world problems, which may involve many global constraints. We explore the use of reformulations that focus on the semantics of high-arity non-binary constraints in our research.

## References

1. Gent, I.P., Stergiou, K., Walsh, T.: Decomposable constraints. In: New Trends in Constraints. (1999) 134–149
2. Bacchus, F., van Beek, P.: On the conversion between non-binary and binary constraint satisfaction problems. In: AAAI/IAAI. (1998) 310–318

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, p. 335, 2002.

# Reformulating Combinatorial Optimization as Constraint Satisfaction

T.K. Satish Kumar

Knowledge Systems Laboratory
Stanford University
tksk@ksl.stanford.edu

## 1 My Research Areas in Brief

I am currently a third year graduate (PhD) student in the Computer Science Department of Stanford University working in the Artificial Intelligence areas of Planning, Scheduling, Diagnosis, Hybrid Systems and Constraint Satisfaction.

## 2 Combinatorial Optimization as Constraint Satisfaction

Constraint satisfaction and combinatorial optimization form the crux of many AI problems. In constraint satisfaction, feasibility-reasoning mechanisms are used to prune the search space, while optimality-reasoning is used for combinatorial optimization. Many AI tasks related to diagnosis, trajectory tracking and planning can be formulated as hybrid problems containing both satisfaction and optimization components, and can greatly benefit from a proper blend of these independently powerful techniques. Sometimes reformulating one to the other helps us to tackle such problems. A good example of this is the fractional packing problem [4]. Here, oracles are defined for a potentially hard problem by replacing a satisfaction component of the problem with the minimization of a certain potential function. Solving a series of these oracles leads us to a solution for the original problem. An instance of the fractional packing problem is the multi-commodity flow problem [3] that is solved using repeated calls to the minimum cost circulation problem (which acts as an oracle).

We proposed the notion of model counting to reformulate combinatorial optimization as constraint satisfaction in [1]. The model counting problem is the problem of estimating the number of solutions to a SAT or a CSP (also referred to as the #SAT problem). The optimization part of a problem can be cast as a search for the right set of constraints that must be satisfied by any good solution. These constraints, which we call the *oracular constraints*, replace the optimization component of a problem to revive the power of constraint reasoning systems that can otherwise be used effectively to solve an unoptimized version of the problem. Reformulating optimization as satisfaction allows us to cast a hybrid problem in satisfaction and optimization as a series of pure constraint satisfaction problems that serve as oracles for the original problem. The oracles

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 336–337, 2002.

are solved making use of the full power of constraint reasoning systems. For example, in the problem of finding the cheapest plan when actions have associated costs, we hope to revive the feasibility reasoning power of Graph-Plan or Black-Box by throwing in additional constraints that act as oracles to the optimization part of the problem.

The number of calls we make to the oracles can be traded off against closeness to the optimal solution for the original problem. In many cases, this continuum is very important and can be exploited to do real-time problem solving. This is because it is often reasonable to find a near-optimal solution within a window of opportunity rather than trying to solve the potentially much harder problem of computing the optimal solution. We are in the process of building a system called SOFTBOX that does this reformulation by making strategic calls to these oracles guided by certain heuristics which we call the *bargain heuristics* (see [1] for details). SOFTBOX is expected to perform better than traditional methods like branch and bound when the satisfaction component of the problem is very hard.

Oracular constraints can also serve as a compact representation for all the reasonably good solutions to a combinatorial optimization problem. A good example of why this can be important is found in trajectory tracking [2]. Here, the best trajectories given the current observations may not be the best once future observations are available. Ideally therefore, we would like to track all the reasonable hypotheses rather than just the best ones (in case the best hypotheses now are ruled out by subsequent observations). The problem is that the number of reasonable candidates can increase over time beyond our computational resources and it becomes impossible to track all of them explicitly. Reformulating the optimization component (posterior probabilities conditioned on observations) associated with the trajectories as satisfaction, we can represent all the good hypotheses at any given point in time as solutions to these oracular constraints. By tracking these constraints rather than the candidates themselves, our representation scales only polynomially with time.

## References

1. Kumar, T. K. S. and Dearden, R. 2002. The Oracular Constraints Method. Proceedings of the Fifth International Symposium on Abstraction, Reformulation and Approximation (SARA 2002).
2. Kurien, J. and Nayak, P. P. 2000. Back to the Future for Consistency-Based Trajectory Tracking. Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000).
3. Leighton, T., Makedon, F., Plotkin, S., Stein, C., Tardos, E. and Tragoudas, S. 1991. Fast Approximation Algorithms for Multicommodity Flow Problem. In Proceedings of the 23rd ACM Symposium on the Theory of Computing, pages 101-111, May 1991.
4. Plotkin, S., Shmoys, D. and Tardos, E. 1995. Fast Approximation Algorithms for Fractional Packing and Covering Problems. Math of Oper. Research, 20(2):257-301, 1995.

# Autonomous Discovery of Abstractions through Interaction with an Environment

Amy McGovern

University of Massachusetts Amherst, Computer Science Department
Amherst, MA, 01003 amy@cs.umass.edu

The ability to create and use abstractions in complex environments, that is, to systematically ignore irrelevant details, is a key reason that humans are effective problem solvers. My research focuses on using machine learning techniques to enable greater autonomy in agents. I am particularly interested in autonomous methods for identifying and creating multiple types of abstractions from an agent's accumulated experience in interacting with its environment. Specific areas of interest include:

- Knowledge representation. How can we efficiently represent the knowledge learned in one task and reuse it for other tasks? This knowledge can take the form of a control policy learned to solve one task or a representation of structure in an environment.
- Autonomous discovery of structure. My dissertation [1] focuses on autonomously identifying and creating useful temporal abstractions from an agent's interaction with its environment.
- Interaction of reinforcement learning and supervised learning methods. I am particularly interested in the combined use of these techniques to create more robust and autonomous learning systems.
- Application of these techniques to robots, with a particular focus on robots assisting a human presence in space.

Although the utility of abstraction is commonly accepted, there has been relatively little research on how to autonomously discover or create useful abstractions. A system that can create new abstractions autonomously can learn and plan in situations that its original designer was not able to anticipate.

My dissertation [1] introduced two related methods that allow an agent to autonomously discover and create temporal abstractions from its accumulated experience with its environment. A *temporal abstraction* is an encapsulation of a complex set of actions into a single higher-level action which allows an agent to learn and plan while ignoring details that appear at finer levels of temporal resolution. The main idea for both methods is to search for patterns that occur frequently within an agent's accumulated successful experience and that do not occur in unsuccessful experiences. These patterns are used to create the new temporal abstractions.

The two types of temporal abstractions that the methods create are 1) subgoals and closed-loop policies for achieving them, and 2) open-loop policies, or simply action sequences, that are useful "macros." [3], [2], and [1] demonstrate the utility of both types of these temporal abstractions in several simulated tasks, including two simulated mobile robot tasks. We demonstrate that the autonomously created temporal abstractions can

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 338–339, 2002.

both facilitate the learning of an agent within a task and can enable effective knowledge transfer to related tasks. As a larger task, we focus on the difficult problem of scheduling the assembly instructions on a pipelined machine in such a manner that the reordered instructions will execute as quickly as possible. We demonstrate that discovered action sequences can significantly improve the performance of the scheduler and can enable effective knowledge transfer across similar processors.

Both methods can extract useful subgoals or sequences from collections of behavioral trajectories generated by different processes. In particular, we demonstrate that the methods can be effective when applied to collections generated by reinforcement learning agents, heuristic searchers, and human tele-operators. These results illustrate that this approach to autonomously generating temporal abstractions is general and can be applied in situations where an autonomous learning agent is not able to solve a task, but where a human or heuristic search agent is able to provide useful feedback.

## References

1. Amy McGovern. *Autonomous Discovery of Temporal Abstractions from Interaction with an Environment*. PhD thesis, University of Massachusetts Amherst, 2002.
2. Amy McGovern and Andrew G. Barto. Accelerating reinforcement learning through the discovery of useful subgoals. In *Proceedings of the 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space: i-SAIRAS 2001*, page electronically published, 2001.
3. Amy McGovern and Andrew G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In C. Brodley and A. Danyluk, editors, *Proceedings of the 18th International Conference on Machine Learning ICML 2001*, pages 361–368, San Francisco, CA, 2001. Morgan Kaufmann Publishers.

# Interface Verification: Discrete Abstractions of Hybrid Systems

Meeko Oishi

Hybrid Systems Laboratory, Stanford University
moishi@stanford.edu

Modern commercial aircraft have extensive automation which helps the pilot fly the aircraft by performing computations, obtaining data, and completing dull tasks. In such a safety-critical system, the pilot display must contain enough information so that the pilot can correctly predict the aircraft's behavior, while not overloading the pilot with unnecessary information. How can we mathematically guarantee that the display provides the pilot with enough information that the pilot can safely and effectively control the aircraft? How can we ensure that the aircraft behaves as the pilot anticipates? My research addresses this issue by extending a theoretical framework, verification of hybrid systems, to include the pilot's interactions with the aircraft automation.

Hybrid systems combine two types of behaviors: how a system evolves over time according to the laws of physics, and how the system evolves according to signals and switches. The combination of continuous and discrete dynamics leads to extremely complex behavior. In the case of the aircraft, this means that we can model how the aircraft flies as well as the logic which drives the aircraft automation. Hybrid systems are controlled through the combination of continuous and discrete signals we can directly alter. Some of these signals are controlled directly by the automation, while other signals are controlled manually by the pilot. While hybrid system verification in a completely automated framework is a well-studied problem, verification of *semi-automated* systems must incorporate how the user interacts with the automation, including what is shown to the user through the interface.

Commercial aircraft require intense validation to certify their dependability and accuracy. Currently, this validation is accomplished through extensive testing of various conditions in simulators and prototypes. *Verification* offers an alternative to this costly process – it provides a mathematical guarantee over all possible conditions and behaviors. Computational tools have been recently developed to aid in the verification of hybrid systems. The hybrid verification methodology not only identifies the "safe" region to operate in, but also constructs the hybrid control necessary to guarantee that the system will never leave the "safe" space. The result is quite powerful – a complex system, subject to real-life errors and limitations, is mathematically guaranteed to be safe in the face of those errors and limitations.

The question my research addresses is: How do we extend this methodology to verify that an interface provides enough information to the user, that the user can safely control the system? We assume the user interacts with the system through the interface, and has a specific task to accomplish.

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 340–341, 2002.

One motivating example is an automatic landing scenario. The aircraft sequences through various modes – holding a constant altitude, descending at a constant rate, then smoothly touching down on the runway – some of which are initiated by the automation, and some of which are initiated by the pilot. If landing is problematic (debris on the runway, imminent collision with another aircraft, excessive wind) the pilot initiates a go-around, a maneuver in which the aircraft quickly climbs to a predetermined altitude. During this high-tempo, safety-critical time of flight, any confusion or unsafe behavior is unacceptable.

The pilot anticipates how the automation will behave based on pilot training, personal experience, the aircraft manual, and information displayed through the cockpit interface. We would like to guarantee that the automation does not surprise the pilot – that the interface provides information that is consistent with how the aircraft actually behaves, and allows the pilot to safely land or safely go-around. The interface abstracts complex behaviors into simple elements: for example, an entire range of continuous behavior is summarized by the sole indication "FLARE" on the pilot's display. Another way to abstract these complicated behaviors arises from the result of our hybrid verification methodology. We compare the result of this abstraction with a discrete model of the interface, according to well-developed discrete user-interface verification techniques.

The result of this comparison tells us whether the interface is adequate for a given task; that is, whether the interface provides the user with enough information that the user can safely and unambiguously control the system. For the automatic landing, it tells us whether the pilot's display provides the pilot with enough information that the pilot can safely land or safely go-around. The main advantage of this methodology is the a priori guarantee that mathematical verification provides. If the interface is adequate, then over *all* possible system behaviors, the user will be able to safely control the system through a desired task. Verification within a hybrid framework allows us to account for the inherently complicated dynamics underlying the simple, discrete representations displayed to the pilot. Incorporating the results of this analysis can help prevent "automation surprises" and procedural mishaps. While verification tools can aid in design, we hope to contribute directly to the design problem, and plan to extend discrete-only interface design techniques to a hybrid framework, as well.

## References

1. M. Oishi, I. Mitchell, A. Bayen, C. Tomlin, and A. Degani. Hybrid verification of an interface for an automatic landing. Submitted to *Proceedings of the IEEE Conference on Decision and Control*, 2002.
2. M. Oishi, C. Tomlin, V. Gopal, and D. Godbole, "Addressing multiobjective control: Safety and performance through constrained optimization," in *Hybrid Systems: Computation and Control* (M. D. Benedetto and A. Sangiovanni-Vincentelli, eds.), LNCS 2034, pp. 459–472, Springer Verlag, March 2001.
3. M. Oishi and C. Tomlin. Switching in nonminimum phase systems: Applications to a VSTOL aircraft. In *Proceedings of the American Control Conference*, June 2000.
4. M. Oishi and C. Tomlin. Switched nonlinear control of a VSTOL aircraft. In *Proceedings of the IEEE Conference on Decision and Control*, December 1999.

# Learning Semi-lattice Codebooks for Image Compression
## (Research Summary)

Yoshiaki Okubo[1] and Xiaobo Li[2]

[1] Division of Electronics and Information Engineering, Hokkaido University
N-13 W-8, Sapporo 060-8628, JAPAN
yoshiaki@db-ei.eng.hokudai.ac.jp
[2] Department of Computing Science, University of Alberta
Edmonton, Alberta, CANADA T6G 2E8
li@cs.ualberta.ca

Our work is concerned with *quantization* in *image compression*. The task of quantization is to approximate transformed data of original images so that the images can be efficiently stored. In previous studies of *abstraction*, *reformulation* and *approximation* (AR&A), the notions seems to be mainly used to improve computational efficiency. Although the purposes might seem to be quite different, an important theme is shared in each case: "*how to create a good* AR&A for a problem setting we are concerned with?"

In lossy image compression, given an image to be compressed, the original pixel values are first mapped into DCT or *wavelet transform* coefficients. Then the coefficients are *quantized* or *approximated*. The performance of the quantizer directly affects the quality of the reconstructed image. We deal with *vector quantization* in our study. In such quantization, each $n$-dimensional coefficient vector of the image is approximated by a *code vector* in a *codebook* defined off-line. The construction of a codebook is critical in designing a good vector quantizer.

There exist many codebook construction methods, including a number of variations of *LBG* [4] algorithm. *LBG* constructs an *explicit* codebook in which every code vector is enumerated. A *lattice vector quantization* [3], on the other hand, defines an *implicit* codebook. In other words, the lattice defines a space packing strategy that a set of $n$-dimensional vectors of certain format forms a *lattice codebook*. In general, a lattice codebook can be defined by a finite set of *base vectors*. Because that the design and search of lattice codebook is usually much faster than *LBG*, and the storage of a lattice requires much smaller space than the explicit vector set in *LBG*, lattice vector quantization has been widely used in image compression (e.g. [1,2]).

At the moment, any lattice quantizer, either vector or scalar, is usually designed to quantize any type of images. However, in real-life, each particular image database usually contains only a limited types of images. For example, a medical chest X-ray database may have images with very high resolution, low-contrast and highly textured pictures with no color. A collection of wildlife photos usually have blue/green (sky/grass) regions that are relatively smooth. In compressing images of a specific type, we might be able to take advantage of the repeated

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 342–343, 2002.

occurrence of certain characteristics of the images and improve the lattice codebook by removing "unnecessary" code vectors and adding additional "useful" code vectors. This codebook is tailored for a particular image type, and can be iteratively updated and modified to fit a certain application.

We are currently studying a learning method that modifies a standard vector codebook to fit one particular image type. We start with a standard lattice codebook that can adequately quantize most input coefficient vectors already. Some coefficient vectors that cannot be approximated by the lattice with admissible noise can be specific to the image type. The codebook is refined to improve the quantization of those vectors.

Such a refinement can be performed in way that is similar to *LBG*. Each "hard-to-handle" vector is treated as a *training vector*. For such a vector $V$, one original lattice code vector which is closest (or the most similar) to $V$ is selected. The closeness can be evaluated by a distance function. The selected code vectors are then used to construct a semi-lattice that better approximates those specific vectors. The additional code vectors are added to the original lattice. Because of the nature of the extra vectors, the resultant codebook is no longer a lattice, thus is called a *semi-lattice codebook*.

Further development of the outlined algorithm is currently under investigation. More details of our experimental results and theoretical analysis will be reported later.

## References

1. J. Knipe, X. Li and B. Han: "An Improved Lattice Vector Quantization Based Scheme for Wavelet Compression", *IEEE Transactions on Signal Processing,* Vol. 46, No. 1, pp. 239–242, 1998.
2. P. Shelley: "New Techniques in Wavelet Image Compression", *Master Thesis, Department of Computing Science, University of Alberta,* 2001.
3. J. H. Conway and N. J. A. Sloane: "Sphere Packings, Lattices and Groups", *Springer-Verlag,* 1988.
4. Y. Linde, A. Buzo and R. Gray: "An Algorithm for Vector Quantizer Design", *IEEE Transactions on Communications,* Vol. COM-28, No. 1, pp. 84–95, 1980.

# Research Summary

Marc Pickett

Autonomous Learning Laboratory
University of Massachusetts, Amherst MA 01002, USA
pickett@cs.umass.edu

Marc's research interests are in how abstractions can be created automatically by a machine learning agent. He is also interested in automatically creating representations, how knowledge can be grounded in sensory information, and methods for bridging the gap between statistical and symbolic computation. More generally, his interests include cognitive neuroscience, and applications of artificial intelligence to understanding complex systems.

Recently, Marc developed an algorithm that makes efficient use of a costly distance metric by using geometric reasoning. This algorithm approximates unknown distances among objects from a relatively small number of known distances. The approximate distances are then used to cluster objects. This technique has been successfully applied to indexing word images of handwritten text. This work has been in collaboration with David Jensen and R. Manmatha [2].

Another recent work has been on how a reinforcement learning agent can create abstract actions (options). His paper on this research presents a method called "PolicyBlocks" by which an agent can create useful options automatically. PolicyBlocks creates options by finding commonly occurring subpolicies from the solutions to a set of sample tasks, then subtracting these subpolicies from the solutions in a manner akin to Principle Components and Explanation Based Learning. Using these options, learning to do future related tasks is accelerated. This increase in performance was illustrated in a "rooms" grid-world, in which the options found by PolicyBlocks outperform even hand designed options, and in a hydroelectric reservoir control task. This work has been in collaboration with Andrew Barto [1].

## References

[2002] Pickett, M., Barto, A.: PolicyBlocks: An Algorithm for Creating Useful Macro-Actions in Reinforcement Learning. in proceedings of the International Conference on Machine Learning (2002)

[2002] Pickett, M., Jensen D., Manmatha, R. Indexing Handwritten Text using Clustering Algorithms. Synthesis Project Report, University of Massachusetts (2002)

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, p. 344, 2002.

# Principled Exploitation of Heuristic Information (Research Summary)

Wheeler Ruml

Division of Engineering and Applied Sciences
Harvard University
33 Oxford Street, Cambridge, MA 02138 USA
ruml@eecs.harvard.edu

My research focuses on combinatorial search and optimization. I am particularly interested in trying to outline general principles that would be of use to practitioners who are confronting a new optimization problem. What are the important characteristics of the problem that will determine which algorithm performs best? My intuition is that the most important questions one should ask about a new problem concern the available sources of heuristic information. It is a truism in AI that knowledge reduces search. Following this line of thinking, I have been focusing on how best to exploit sources of heuristic information.

My most recent work concerns tree search for combinatorial optimization and constraint satisfaction. In these problems, one seeks the minimum cost leaf in a tree of bounded depth. Most leaves will lie at similar depths, not far from the depth bound. This scenario encompasses both traditional combinatorial optimization problems such as the traveling salesman problem as well as constraint satisfaction problems, for which the cost function is the number of violated constraints. There are several sources of domain-specific heuristic information: the variable choice heuristic, the value choice heuristic, and the solution costs themselves. A true solution to these problems will include a way to use and integrate all of these sources of information.

I am particularly interested in anytime methods that explicitly attempt to adapt on-line to the current problem instance via flexible search control and efficient learning and metareasoning. An anytime algorithm is particularly important in real-time settings, where proving that a solution is optimal by completely enumerating the search space is infeasible and we merely would like the best solution we can find in the allotted time. Almost all real-world problems fall into this category.

A first step toward an algorithm that incorporates these ideas for tree search is given in Ruml (2002), building on previous work reported in Ruml (2001a). The algorithm, *best-leaf-first search*, is based on on-line estimation of a model of the search space. This model is then used to guide the search, visiting leaves in increasing predicted cost. This is the rational order with respect to the algorithm's current model. The search actually uses multiple expanding sweeps of the space, reminiscent of iterative deepening, so it is not a problem that the model is being refined on-line. Multiple sources of information can easily be incorporated into the model. The framework is very general, and should apply to most combinatorial optimization and constraint satisfaction systems. My dissertation,

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, pp. 345–346, 2002.

to be completed in May of 2002, contains more details (Ruml, in preparation). Other relevant early work of mine includes Ruml (2001c) and Ruml (2001b).

One important future direction is to take a similar approach, emphasizing principled exploitation of heuristic information and explicit learning, to improvement-based search (as opposed to tree search). I have made a preliminary attempt in the setting in which one has gradient information available, by which I mean information about which variable's change might lead to the greatest immediate improvement. This information is available in constraint satisfaction problems such as boolean satisfiability and is used by algorithms such as WalkSAT. I have experimented with a hill-climbing algorithm that takes into account the idea that good local minima are likely to cluster. This leads to a variation of the classic Kernighan-Lin search method. Results on a combinatorial machine learning problem are discussed briefly in Ruml (to appear).

## References

Ruml, Wheeler. 2001a. Incomplete tree search using adaptive probing. In *Proceedings of IJCAI-01*, pages 235–241.

Ruml, Wheeler. 2001b. Stochastic tree search: Where to put the randomness? In Holger H. Hoos and Thomas G. Stützle, editors, *Proceedings of the IJCAI-01 Workshop on Stochastic Search*, pages 43–47.

Ruml, Wheeler. 2001c. Using prior knowledge with adaptive probing. In Carla Gomes and Toby Walsh, editors, *Proceedings of the 2001 AAAI Fall Symposium on Using Uncertainity Within Computation*, pages 116–120. AAAI Technical Report FS-01-04.

Ruml, Wheeler. 2002. Heuristic search in bounded-depth trees: Best-leaf-first search. Technical Report TR-01-02, Harvard University, Cambridge, MA, January.

Ruml, Wheeler. in preparation. *Adaptive Tree Search*. Ph.D. thesis, Harvard University.

Ruml, Wheeler. to appear. Constructing distributed representations using additive clustering. In *Advances in Neural Information Processing Systems 14 (NIPS-01)*. MIT Press.

# Reformulation of Temporal Constraint Networks

Lin Xu

Constraint Systems Laboratory
Department of Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln NE, 68588-0115
lxu@cse.unl.edu

The focus of our research is the design and evaluation of new reformulation techniques for temporal constraint networks. The general temporal constraint satisfaction problem (TCSP) is NP-hard [2]. Although a simplified version of it, the simple temporal problem (STP), is tractable [2], techniques are being sought to further reduce the computational effort needed in practice in order to handle time critical conditions. One example is planning of the activities of an autonomous remote agent where some desirable properties such as dispatchability can be guaranteed.

We have experimented with three techniques to reduce the amount of necessary constraint propagation. These are techniques that exploit the triangulation of constraint graph [1], decompose the network into bi-connected components [3], and test the non-existence of cycles to the graph. We have evaluated these techniques on problems generated by a random problem generator we have designed that guarantees the existence of at least one solution. We have shown that significant gains can be obtained and the performance is systematically improved.

In our future investigations we will design and evaluate new reformulation techniques that exploit the structure of the constraint network and the semantics of the temporal constraints in order to enhance the performance of problem solving and guarantee some properties such as robustness and dispatchability, which are desirable in practice.

## References

1. Christian Bliek and Djamilla Sam-Haroud. Path Consistency for Triangulated Constraint Graphs. In *Proc. of the 16 th IJCAI*, pages 456–461, Stockholm, Sweden, 1999.
2. Rina Dechter, Itay Meiri, and Judea Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49:61–95, 1991.
3. Eugene C. Freuder. A Sufficient Condition for Backtrack-Bounded Search. *JACM*, 32 (4):755–761, 1985.

S. Koenig and R. Holte (Eds.): SARA 2002, LNAI 2371, p. 347, 2002.

# Author Index