

Ian Miguel
Wheeler Ruml (Eds.)

LNAI 4612

Abstraction, Reformulation, and Approximation

7th International Symposium, SARA 2007
Whistler, Canada, July 2007
Proceedings

 Springer

Lecture Notes in Artificial Intelligence 4612

Edited by J. G. Carbonell and J. Siekmann

Subseries of Lecture Notes in Computer Science

Ian Miguel Wheeler Ruml (Eds.)

Abstraction, Reformulation, and Approximation

7th International Symposium, SARA 2007
Whistler, Canada, July 18-21, 2007
Proceedings

Series Editors

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

Volume Editors

Ian Miguel
University of St. Andrews
School of Computer Science
North Haugh, KY16 9SX, St. Andrews, UK
E-mail: ianm@cs.st-and.ac.uk

Wheeler Ruml
Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto, CA 94304, USA
E-mail: ruml@acm.org

Library of Congress Control Number: 2007930461

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

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN 0302-9743
ISBN-10 3-540-73579-8 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-73579-3 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media

springer.com

© Springer-Verlag Berlin Heidelberg 2007
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12089598 06/3180 5 4 3 2 1 0

Preface

It has been recognized since the inception of artificial intelligence 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 in a variety of problem-solving settings, including automated reasoning, cognitive modelling, constraint programming, design, diagnosis, machine learning, model-based reasoning, planning, reasoning, scheduling, search, theorem proving, and intelligent tutoring. The primary use of AR&A techniques in such settings has been to overcome computational intractability by decreasing the combinatorial costs associated with searching large spaces. In addition, AR&A techniques are useful for knowledge acquisition and explanation generation in complex domains.

The considerable interest in AR&A techniques has led to a series of successful symposia over the last decade, the Symposium on Abstraction, Reformulation, and Approximation (SARA). Its aim is to provide a forum for intensive interaction among researchers in all areas of artificial intelligence and computer science interested in the different aspects of AR&A. AAAI workshops in 1990 and 1992 focused on selecting, constructing, and using abstractions and approximations, while a series of workshops in 1988, 1990, and 1992 focused on problem reformulations. The two series were then combined since there was considerable overlap in their attendees and topics. The present symposium is the seventh in this new series, following successful symposia in 1994, 1995, 1998, 2000, 2002, and 2005. The diverse backgrounds of participants of the symposia lead to a rich and lively exchange of ideas, allow the comparison of goals, techniques, and paradigms between researchers who might not otherwise be aware of each others' work, and help identify important research issues and engineering hurdles.

This volume contains the proceedings of SARA 2007, the seventh symposium, held at Whistler Village, British Columbia, Canada, July 18-21. Three distinguished speakers were invited to give keynote presentations, and their abstracts are included herein: Vadim Bulitko of the University of Alberta, Canada; Alan Frisch of the University of York, UK; and John Hooker of Carnegie Mellon University, USA. We thank all the authors of submitted papers for their efforts in preparing an impressive corpus of work, the Programme Committee and auxiliary reviewers for their thorough evaluation and considered selection of presentations for SARA, and the SARA Steering Committee for its advice and guidance. We are also very grateful to Google, the Pacific Institute for the Mathematical Sciences, and the Palo Alto Research Center for their generous support of the symposium, and to the Association for the Advancement of Artificial Intelligence (AAAI), with which SARA 2007 is affiliated.

July 2007

Ian Miguel
Wheeler Ruml

Organization

Steering Committee

Berthe Choueiry (University of Nebraska-Lincoln, USA)
Fausto Giunchiglia (University of Trento, Italy)
Michael Genesereth (Stanford University, USA)
Robert Holte (University of Alberta, Canada)
Ian Miguel (University of St. Andrews, UK)
Michael Lowry (NASA, USA)
Wheeler Ruml (Palo Alto Research Center, USA)
Lorenza Saitta (Università del Piemonte Orientale, Italy)
Sven Koenig (University of Southern California, USA)
Toby Walsh (University of New South Wales, Australia)

Organizing Committee

Conference Chairs	Ian Miguel (University of St. Andrews, UK) Wheeler Ruml (Palo Alto Research Center, USA)
Publicity Chair	Karen Petrie (University of Oxford, UK)
Sponsorship Chair	Peter Nightingale (University of St. Andrews, UK)

Programme Committee

J. Christopher Beck (University of Toronto, Canada)
Berthe Choueiry (University of Nebraska-Lincoln, USA)
Johan de Kleer (Palo Alto Research Center, USA)
Marie desJardins (University of Maryland, Baltimore County, USA)
Stefan Edelkamp (Universität Dortmund, Germany)
Boi Faltings (Ecole Polytechnique Federale de Lausanne, Switzerland)
Ariel Felner (Ben-Gurion University, Israel)
Alan Frisch (University of York, UK)
Hector Geffner (Universitat Pompeu Fabra, Spain)
Michael Genesereth (Stanford University, USA)
Fausto Giunchiglia (University of Trento, Italy)
Brahim Hnich (Izmir University, Turkey)
Daniel Kayser (Université Paris-Nord, France)
Sven Koenig (University of Southern California, USA)
Derek Long (University of Strathclyde, UK)
Michael Lowry (NASA, USA)

Peter Revesz (University of Nebraska-Lincoln, USA)
Marie-Christine (Rousset University of Grenoble, France)
Lorenza Saitta (Università del Piemonte Orientale, Italy)
Bart Selman (Cornell University, USA)
Barbara Smith (Cork Constraint Computation Centre, Ireland)
Miroslav Velev (Carnegie Mellon University, USA)
Toby Walsh (University of New South Wales, Australia)
Weixiong Zhang (Washington University, USA)
Rong Zhou (Palo Alto Research Center, USA)
Robert Zimmer (Goldsmiths College, University of London, UK)
Jean-Daniel Zucker (Université Paris 13/UR 079 GEODES, France)

Additional Referees

Sebastian Brand	Peter Nightingale	Xiaoming Zheng
Kenneth Daniel	Claude-Guy Quimper	
George Katsirelos	Pietro Torasso	
Zeynep Kiziltan	Gianluca Torta	

Sponsoring Institutions

The Association for the Advancement of Artificial Intelligence
Google, Inc.
The Pacific Institute for the Mathematical Sciences
The Palo Alto Research Center

Table of Contents

Invited Talks (Abstracts)

State Abstraction in Real-Time Heuristic Search	1
<i>Christoph Helm</i>	
Abstraction and Reformulation in the Generation of Constraint Models	2
<i>Stefan Edelkamp</i>	
A Framework for Integrating Optimization and Constraint Programming.....	4
<i>Christoph Helm</i>	

Research Papers

DFS-Tree Based Heuristic Search	5
<i>Christoph Helm, Stefan Edelkamp, and Alexander Schödl</i>	
Partial Pattern Databases	20
<i>Christoph Helm, Stefan Edelkamp, and Alexander Schödl</i>	
CDB-PV: A Constraint Database-Based Program Verifier	35
<i>Christoph Helm, Stefan Edelkamp, and Alexander Schödl</i>	
Generating Implied Boolean Constraints Via Singleton Consistency.....	50
<i>Christoph Helm</i>	
Reformulating Constraint Satisfaction Problems to Improve Scalability	64
<i>Christoph Helm, Stefan Edelkamp, and Alexander Schödl</i>	
Reformulating Global Constraints: The SLIDE and REGULAR Constraints	80
<i>Christoph Helm, Stefan Edelkamp, and Alexander Schödl</i>	
Relaxation of Qualitative Constraint Networks.....	93
<i>Christoph Helm, Stefan Edelkamp, and Alexander Schödl</i>	
Dynamic Domain Abstraction Through Meta-diagnosis	109
<i>Christoph Helm</i>	

Channeling Abstraction	124
Approximate Model-Based Diagnosis Using Greedy Stochastic Search	139
Combining Perimeter Search and Pattern Database Abstractions	155
Solving Satisfiability in Ground Logic with Equality by Efficient Conversion to Propositional Logic	169
Tailoring Solver-Independent Constraint Models: A Case Study with ESSENCE' and MINION	184
A Meta-CSP Model for Optimal Planning	200
Reformulation for Extensional Reasoning	215
An Abstract Theory and Ontology of Motion Based on the Regions Connection Calculus	230
Computing and Using Lower and Upper Bounds for Action Elimination in MDP Planning	243
Model-Based Exploration in Continuous State Spaces	258
Active Learning of Dynamic Bayesian Networks in Markov Decision Processes	273
Boosting MUS Extraction	285
Homogeneous Hierarchical Composition of Areas in Multi-robot Area Coverage	300
Formalizing the Abstraction Process in Model-Based Diagnosis	314
Boolean Approximation Revisited	329

An Analysis of Map-Based Abstraction and Refinement	344
Solving Difficult SAT Instances Using Greedy Clique Decomposition	359
Abstraction and Complexity Measures	375
Research Summaries	
Abstraction, Emergence, and Thought	391
What's Your Problem? The Problem of Problem Definition	393
A Reformulation-Based Approach to Explanation in Constraint Satisfaction	395
Integration of Constraint Programming and Metaheuristics	397
Rule-Based Reasoning Via Abstraction	399
Extensional Reasoning	400
Reformulating Constraint Models Using Input Data	402
Using Analogy Discovery to Create Abstractions	405
Distributed CSPs: Why It Is Assumed a Variable per Agent?	407
Decomposition of Games for Efficient Reasoning	409
Generalized Constraint Acquisition	411
Using Infeasibility to Improve Abstraction-Based Heuristics	413
Leveraging Graph Locality Via Abstraction	415
Author Index	417

State Abstraction in Real-Time Heuristic Search

Vadim Bulitko

University of Alberta, ****, Canada
bulitko@ualberta.ca

Abstract. Real-time heuristic search methods, such as LRTA*, are used by situated agents in applications that require the amount of planning per move to be independent of the problem size. Such agents plan only a few actions at a time in a local search space and avoid getting trapped in local minima by improving their heuristic function over time. In this talk we present recent extensions to LRTA* based on automated state abstraction – an idea that has proved powerful in other areas of search and learning. In one of the extensions, learning performance of LRTA* is improved by running it in a smaller abstract search space. The resulting algorithm retains real-time performance and completeness/ convergence properties. Empirically, the abstraction is found to improve efficiency by trading off planning time, learning speed and other antagonistic performance measures. The talk will be illustrated with applications to path-planning in computer video games.

Abstraction and Reformulation in the Generation of Constraint Models*

(Extended Abstract)

Alan M. Frisch

University of York, York YO10 5DD, UK
frisch@cs.york.ac.uk

Many and diverse combinatorial problems have been solved with great success using constraint programming. However, to employ constraint programming technology to solve a problem, the problem first must be characterised, or, more precisely, by a set of constraints that its solutions must satisfy. Generating a correct model can be difficult; generating one that is easier to solve than its alternatives is even more difficult, often requiring considerable expertise. This so-called “modelling bottleneck” has inhibited the wider use of constraint programming technology.

Our work has addressed this modelling bottleneck by designing a rule-based system, called CONJURE, that automatically generates constraint models by reformulating problem specifications given in an abstract language, called ESSENCE. This talk introduces and discusses the design of CONJURE and ESSENCE, emphasising the central roles of abstraction and reformulation.

ESSENCE is a new language for specifying combinatorial (decision or optimisation) problems at a high level of abstraction. It is the result of our attempt to design a formal language that enables problem specifications that are similar to rigorous specifications that use a mixture of natural language and discrete mathematics, such as those catalogued by Garey and Johnson [1]. Its most important facility for abstraction is the provision of decision variables whose values can be combinatorial objects, such as tuples, sets, multisets, relations, partitions and functions. ESSENCE also allows these combinatorial objects to be nested to arbitrary depth, thus providing, for example, sets of partitions, sets of sets of partitions, and so forth. Thus, a problem that requires finding a combinatorial object of a certain type can be specified directly in the language by using a decision variable of that type, thereby eliminating the need to model the object by a collection of variables of simple types.

Using a set of recursive rules, CONJURE can reformulate a problem specified in ESSENCE into a set of alternative constraint models. Each of these is a correct model of the problem formulated at a level of abstraction supported by existing constraint toolkits. The development of CONJURE required designing reformulation rules whose recursive structure could handle the nested types of ESSENCE.

* The work report here was done in collaboration with Matthew Grum, Chris Jefferson, Bernadette Martínez Hernández and Ian Miguel.

This talk explains why this is a difficult problem and how we ultimately solved it.

Further information on ESSENCE and CONJURE can be found within [2] and [3] and at <http://www.cs.york.ac.uk/aig/constraints/AutoModel>.

References

1. Garey, M.R., Johnson, D.S.: Computers and Intractability. W. H. Freeman, New York (1979)
2. Frisch, A.M., Grum, M., Jefferson, C., Hernández, B.M., Miguel, I.: The design of Essence: A language for specifying combinatorial problems. In: Proc. of the Twentieth Int. Joint Conf. on Artificial Intelligence (2007)
3. Frisch, A.M., Jefferson, C., Hernández, B.M., Miguel, I.: The rules of constraint modelling. In: Proc. of the Nineteenth Int. Joint Conf. on Artificial Intelligence, pp. 109–116 (2005)

A Framework for Integrating Optimization and Constraint Programming

J.N. Hooker

Carnegie Mellon University, USA
john@hooker.tepper.cmu.edu

Abstract. This talk begins with a description of the modeling and computational advantages that can be obtained by combining optimization and constraint programming in a principled way. It then presents a framework for integration based on three elements: a search-infer-and-relax algorithmic paradigm, a unifying theory of duality, and the use of meta-constraints (a generalization of global constraints) for modeling. Inference techniques from constraint programming and relaxation techniques from mathematical programming are combined in both branch-and-relax search and constraint-based (nogood-based) search. The talk illustrates these ideas with examples in freight shipment, employee scheduling, continuous global optimization, airline crew scheduling, the propositional satisfiability problem, and multiple machine scheduling.

DFS-Tree Based Heuristic Search^{*}

Montserrat Abril, Miguel A. Salido, and Federico Barber

Dpt. of Information Systems and Computation, Technical University of Valencia
Camino de Vera s/n, 46022, Valencia, Spain
{mabril, msalido, fbarber}@dsic.upv.es

Abstract. In constraint satisfaction, local search is an incomplete method for finding a solution to a problem. Solving a general constraint satisfaction problem (CSP) is known to be NP-complete; so that heuristic techniques are usually used. The main contribution of this work is twofold: (i) a technique for de-composing a CSP into a DFS-tree CSP structure; (ii) an heuristic search technique for solving DFS-tree CSP structures. This heuristic search technique has been empirically evaluated with random CSPs. The evaluation results show that the behavior of our heuristic outperforms than the behavior of a centralized algorithm.

Keywords: Constraint Satisfaction Problems, CSP Decomposition, heuristic search, DFS-tree.

1 Introduction

One of the research areas in computer science that has gained increasing interest during last years is constraint satisfaction, mainly because many problems like planning, reasoning, diagnosis, decision support, scheduling, etc., can be formulated as constraint satisfaction problems (CSPs) and can be solved by using constraint programming techniques in an efficient way.

Many techniques to solve CSPs have been developed; some originate from solving other types of problems and some are specifically for solving CSPs. Basic CSP solving techniques include: search algorithms, problem reduction, and heuristic strategies.

The more basic sound and complete search technique is Chronological Backtracking. It systematically traverses the entire search space in a depth-first manner. It instantiates one variable at a time until it either finds a solution or proves no solutions exist. However, it can be inefficient because of thrashing. To improve efficiency during search, some basic stochastic algorithms have been developed. Random guessing algorithm is the most naive stochastic search. Like blindly throwing darts, it repeatedly 'guesses' a complete assignment and checks if the assignment satisfies the constraints until it finds a solution or reaches timeout

^{*} This work has been partially supported by the research projects TIN2004-06354-C02-01 (Min. de Educacion y Ciencia, Spain-FEDER), FOM-70022/T05 (Min. de Fomento, Spain), GV/2007/274 (Generalidad Valenciana) and by the Future and Emerging Technologies Unit of EC (IST priority - 6th FP), under contract no. FP6-021235-2 (project ARRIVAL).

(or some maximum number of iterations). Since the algorithm assigns variables in a non-systematic way, it neither avoids checking for the same assignment repeatedly, nor guarantees to verify all possible assignments. Because it does not guarantee to check all possible assignments, the algorithm is incomplete and so it cannot guarantee a solution or prove no solution exists.

However, searching for solutions can be very time consuming, especially if the search space is large and the solutions are distributed in a haphazard way. To improve the efficiency, one can sometimes trim the size of the search space and simplify the original problem. Problem reduction [11] is such a method that can be used at the beginning of a search or during search. Once a problem becomes smaller and simpler, search algorithms can go through the space faster. In some cases, problem reduction can solve CSPs without searching [11].

Because neither basic search nor consistency checks alone can always solve CSPs in a timely manner, adding heuristics and using hybrid algorithms are often used to improve efficiency. For instance, a common strategy is to vary the order in which variables, domain values and constraint are searched. Some algorithms incorporate features such as ordering heuristics, (*variable ordering* and *value ordering* [7], and constraint ordering [12], [8]). Furthermore, some stochastic local search techniques have also been developed for solving CSPs (tabu search, iterated local search, ant colony, etc. [10], [9]).

Furthermore, many researchers are working on graph partitioning [4], [6]. The main objective of graph partitioning is to divide the graph into a set of regions such that each region has roughly the same number of nodes and the sum of all edges connecting different regions is minimized. Achieving this objective is a hard problem, although many heuristic may solve this problem efficiently. For instance, graphs with over 14000 nodes and 410000 edges can be partitioned in under 2 seconds [5]. Therefore, we can apply graph partitioning techniques to decompose a binary CSP into semi-independent sub-CSPs.

In this paper, we present a method for structuring and solving binary CSPs. To this end, first, the binary CSP is decomposed into a DFS-tree CSP structure, where each node represents a subproblem. Then, we introduce a heuristic search algorithm which carries out the search in each node according to the partial solution of parent node and the pruning information of children nodes.

In the following section, we present some definitions about CSPs and classify them in three categories. A method of decomposition into a DFS-tree CSP structure is presented in section 3. A heuristic search technique for solving the DFS-tree CSP structure is presented in section 4. An evaluation of our heuristic search technique is carried out in section 5. Finally, we summarize the conclusions in section 6.

2 Centralized, Distributed and Decomposed CSPs

In this section, we present some basic definitions related to CSPs.

A **CSP** consists of: a set of variables $X = \{x_1, \dots, x_n\}$; each variable $x_i \in X$ has a set D_i of possible values (its domain); a finite collection of constraints $C = \{c_1, \dots, c_p\}$ restricting the values that the variables can simultaneously take.

A **solution** to a CSP is an assignment of values to all the variables so that all constraints are satisfied; a problem with a solution is termed *satisfiable* or *consistent*.

A **binary constraint network** is one in which every constraint subset involves at most two variables. In this case the network can be associated with a constraint graph, where each node represents a variable, and the arcs connect nodes whose variables are explicitly constrained [1].

A **DFS-tree CSP structure** is a tree whose nodes are composed by subproblems, where each subproblem is a CSP (sub-CSPs). Each node of the DFS-tree CSP structure is a **DFS-node** and each individual and atomic node of each sub-CSP is a **single-node**; each *single-node* represents a variable, and each *DFS-node* is made up of one or several *single-nodes*. Each constraint between two *single-nodes* of different *DFS-nodes* is called **inter-constraint**. Each constraint between two *single-nodes* of the same *DFS-node* is called **intra-constraint**.

Partition : A partition of a set C is a set of disjoint subsets of C whose union is C . The subsets are called the blocks of the partition.

Distributed CSP: A distributed CSP (DCSP) is a CSP in which the variables and constraints are distributed among automated agents [13].

Each agent has a set of variables; it knows the domains of its variables and a set of *intra-constraints*, and it attempts to determine the values of its variables. However, there are *inter-constraints* and the value assignment must also satisfy these *inter-constraints*.

2.1 Decomposition of CSPs

There exist many ways for solving a CSP. However, we can classify these problems into three categories: Centralized problems, Distributed problems and Decomposable problems.

- A CSP is a *centralized CSP* when there is no privacy/security rules between parts of the problem and all knowledge about the problem can be gathered into one process. It is commonly recognized that centralized CSPs must be solved by centralized CSP solvers. Many problems are represented as typical examples to be modelled as a centralized CSP and solved using constraint programming techniques. Some examples are: sudoku, n-queens, map coloring, etc.
- A CSP is a *distributed CSP* when the variables, domains and constraints of the underlying network are inherently distributed among agents. This distribution is due to entities identified in the problem (which group a set of variables and constraints among them), constraints may be strategic information that should not be revealed to competitors, or even to a central authority; a failure of one agent can be less critical and other agents might be able to find a solution without the failed agent. Examples of such systems are sensor networks, meeting scheduling, web-based applications, etc.

- A CSP is a *decomposable CSP* when the problem can be divided into smaller problems (subproblems) and a coordinating (master-) entity. For example, the search space of a CSP can be decomposed into several regions and a solution could be found by using parallel computing.

Note that centralized and distributed problems are inherent features of problems. Therefore, a distributed CSP can not be solved by a centralized technique. However, can be solved an inherently centralized CSP by a distributed technique? The answer is 'yes' if we previously decompose the CSP.

Usually, real problems imply models with a great number of variables and constraints, causing dense network of inter-relations. This kind of problems can be handled as a whole only at overwhelming computational cost. Thus, it could be an advantage to decompose this kind of problems to several simpler interconnected sub-problems which can be more easily solved.

In the following example we show that a centralized CSP could be decomposed into several subproblems in order to obtain simpler sub-CSPs. In this way, we can apply a distributed technique to solve the decomposed CSP.

The map coloring problem is a typically centralized problem. The goal of a map coloring problem is to color a map so that regions sharing a common border have different colors. Let's suppose that we must to color each country of Europe. In Figure 1 shows a colored portion of Europe. This problem can be solved by a centralized CSP solver. However, if the problem is to color each region of each country (Spain, Figure 3; France, Figure 4) of Europe, it is easy to think that the problem can be decomposed into a set of subproblems, grouped by clusters). This problem can be solved as a distributed problem, even when the problem is not inherently distributed.

A map coloring problem can be solved by first converting the map into a graph where each region is a vertex, and an edge connects two vertices if and only if the corresponding regions share a border. In our problem of coloring the regions of each country of Europe, it can be observed that the corresponding graph maintains clusters (Spain, Figure 3; France, Figure 4) representing each country. Thus, the problems can be solved in a distributed way.

Following, we present a technique for i) decomposing a binary CSP into several sub-CSPs and ii) structuring the obtained sub-CSPs into a DFS-tree CSP structure. Then, in section 4, we propose a heuristic search technique for solving DFS-tree CSP structures.

3 How to Decompose a Binary CSP into a DFS-Tree CSP Structure

Given any binary CSP, it can be translated into a DFS-tree CSP structure. However, there exist many ways to decompose a graph into a DFS-tree. Depending on the user requirements, it may be desirable to obtain balanced DFS-nodes, that is, each DFS-node maintains roughly the same number of single-nodes; or it may be desirable to obtains DFS-nodes in such a way that the number of edges connecting two single-trees is minimized.

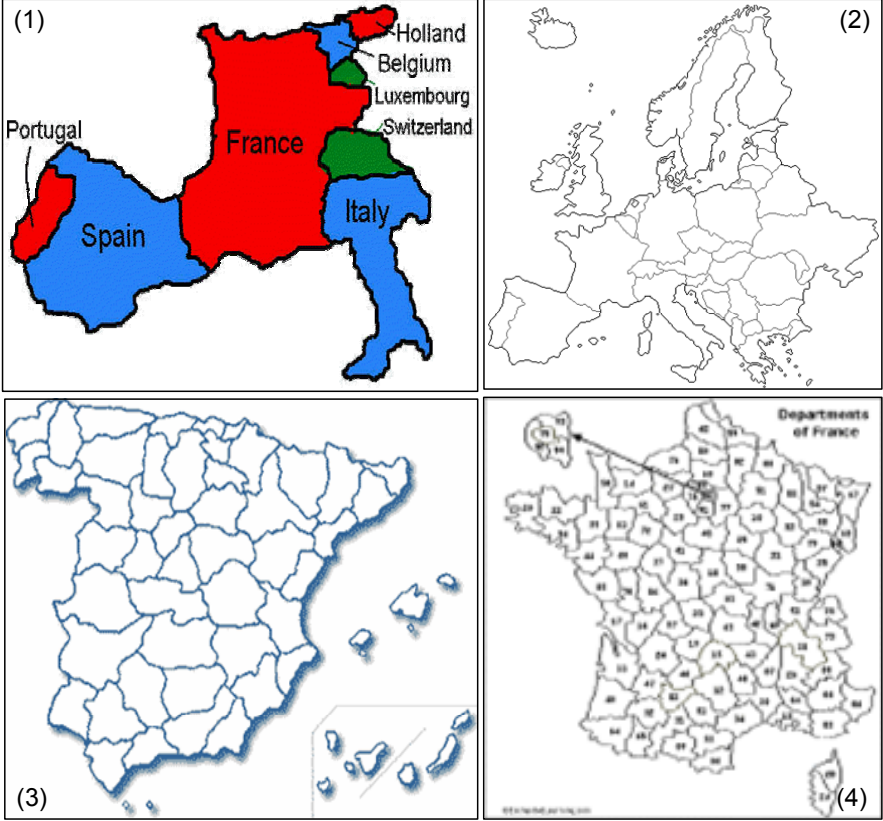


Fig. 1. Map coloring of Europe

We present a proposal which is focused on decomposing the problem by using graph partitioning techniques. Specifically, the problem decomposition is carried out by means of a graph partitioning software called METIS [5]. METIS provides two programs *pmetis* and *kmetis* for partitioning an unstructured graph into k roughly equal partitions, such that the number of edges connecting nodes in different partitions is minimized. We use METIS to decompose a CSP into several sub-CSPs so that *inter-constraints* among variables of each sub-CSP are minimized. Each *DFS-node* will be composed by a sub-CSP.

The next step is to build the DFS-tree CSP structure with k *DFS-nodes* in order to be studied by *agents*. This DFS-tree CSP structure is used as a hierarchy to communicate messages between *DFS-nodes*. The DFS-tree CSP structure is built using Algorithm 1. The nodes and edges of graph G are respectively the *DFS-nodes* and *inter-constraints* obtained after the CSP decomposition. The root *DFS-node* is obtained by selecting the most constrained *DFS-node*. DFS-Structure algorithm then simply put *DFS-node* v into DFS-tree CSP structure (*process(v)*), initializes a set of markers so we can tell which vertices are visited,

chooses a new *DFS-node* i , and calls recursively $\text{DFSStructure}(i)$. If a *DFS-node* has several adjacent *DFS-nodes*, it would be equally correct to choose them in any order, but it is very important to delay the test for whether a *DFS-nodes* is visited until the recursive calls for previous *DFS-nodes* are finished.

Algorithm DFSStructure(G, v)

Input: Graph G , originally all nodes are unvisited. Start DFS-node v of G

Output: DFS-Tree CSP structure

process(v);

mark v as visited;

forall *DFS-node* i adjacent¹ to v not visited **do**
 DFSStructure(i);

end

/* (1) *DFS-node* i is adjacent to *DFS-node* v if at least one
 inter-constraint exists between i and v . */

Algorithm 1. DFSStructure Algorithm

3.1 Example of DFS-Tree CSP Structure

Figure 2 shows two different representation of an example of CSP generated by a random generator module $generate_R(C, n, k, p, q)$, where C is the constraint network; n is the number of variables in network; k is the number of values in each of the domains; p is the probability of a non-trivial edge; q is the probability of an allowable pair in a constraint. This figure represents the constraint network $\langle C, 20, 50, 0.1, 0.1 \rangle$. This problem is very hard to solved with well-known CSP solver methods: Forward-Checking (FC) and Backjumping (BJ). We can see that this problem can be divided into several cluster (see Figure 3) and it can be converted into a DFS-tree CSP structure (see Figure 3). By using our DFS-tree heuristic, it is solved in less than one seconds, and by using FC, this problem has not been solved in several minutes.

In Figure 4 we can see a specific sequence of nodes (a numeric order). Following this sequence, FC algorithm has a great drawback due to the variables without in-links (link with a previous variable): 1, 2, 3, 6, 7, 8, 9, 10, 11 and 16 (see Figure 4). Theses variables have not bounded domains, thus provoking the exploration of all their domains when the algorithm backtracks. This kind of variables is an extreme case of variables with their domain weakly bounded. An example of this situation can be seen in Figure 4, where the variable 12 has its domain bounded by the variable 6. When the bounded domain of the variable 12 has not a valid assignment, FC algorithm backtracks to change the values of the bounded domain, but it will need examine completely the domain of variables 11, 10, 9, 8 and 7 before it changes the assignment of variable 6. In this example, with *domain size* = 50, it involves 50^5 assignments in vain.

¹ A library of routines for experimenting with different techniques for solving binary CSPs is available at <http://ai.uwaterloo.ca/~vanbeek/software/software.html>

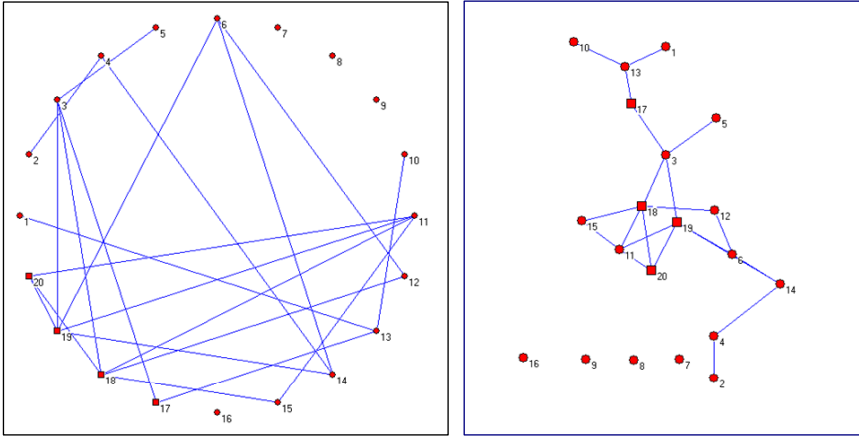


Fig. 2. Random CSP $\langle n = 20, d = 50, p = 0.1, q = 0.1 \rangle$

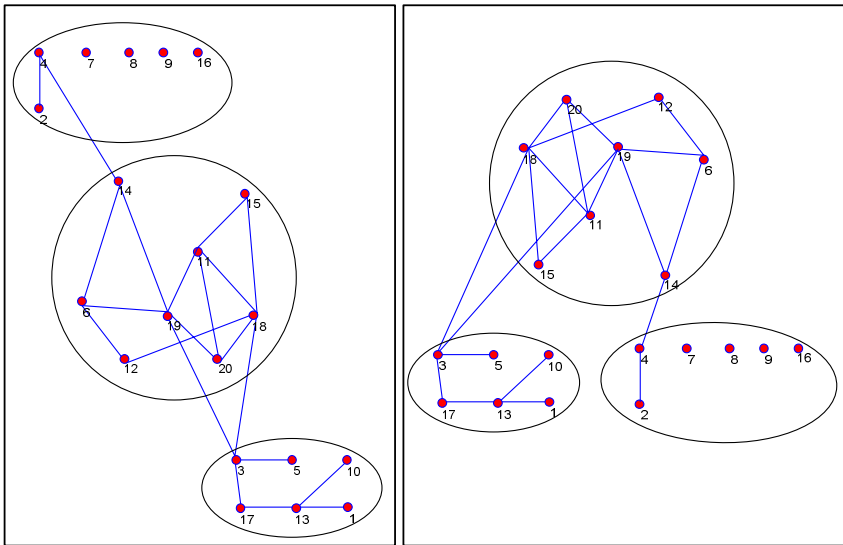


Fig. 3. Left: Decomposed Problem. Right: DFS-Tree CSP structure.

4 DFS-Tree Based Heuristic Search (DTH)

In section [3](#) we have presented a method for structuring a binary CSP into a DFS-Tree CSP structure. In this section we propose a new heuristic search technique for solving DFS-Tree CSP structures.

Our Heuristic called DFS-Tree Based Heuristic Search (DTH) can be considered as a distributed and asynchronous technique. In the specialized literature, there are many works about distributed CSPs. In [\[13\]](#), Yokoo et al. present a

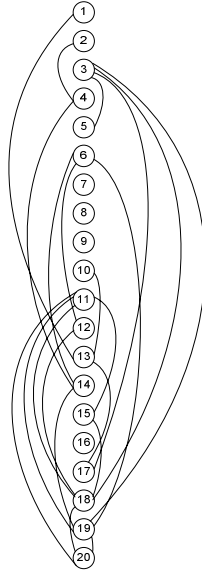


Fig. 4. Sequence of nodes for Forward-Checking Algorithm

formalization and algorithms for solving distributed CSPs. These algorithms can be classified as either centralized methods, synchronous or asynchronous backtracking [13].

DTH is committed to solve the DFS-tree CSP structure in a *Depth-First Search Tree (DFS Tree)* where the root *DFS-node* is composed by the most constrained sub-CSP, in the sense that this sub-CSP maintains a higher number of single-nodes. DFS trees have already been investigated as a means to boost search [2]. Due to the relative independence of nodes lying in different branches of the *DFS tree*, it is possible to perform search in parallel on these independent branches.

Once the variables are divided and arranged, the problem can be considered as a distributed CSP, where a group of *agents* manages each sub-CSP with its variables (single-nodes) and its constraints (edges). Each *agent* is in charge of solving its own sub-CSP by means of a search. Each subproblem is composed by its CSP subject to the variable assignment generated by the ancestor *agents* in the *DFS-tree CSP structure*.

Thus, the root *agent* works on its subproblem (root meta-node). If the root *agent* finds a solution then it sends the consistent partial state to its children *agents* in the *DFS-tree*, and all children work concurrently to solve their specific subproblems knowing consistent partial states assigned by the root *agent*. When a child *agent* finds a consistent partial state it sends again this partial state to its children and so on. Finally, leaf *agents* try to find a solution to its own subproblems. If each leaf *agent* finds a consistent partial state, it sends an OK

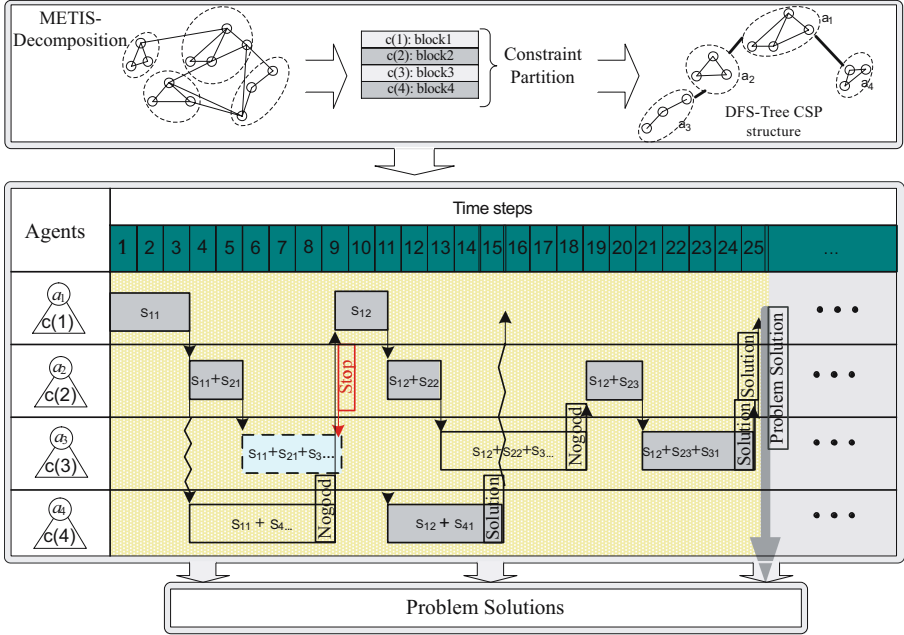


Fig. 5. Decomposing Technique and DFS-Tree Based Heuristic Search

message to its parent *agent*. When all leaf *agents* answer with OK messages to their parents, a solution to the entire problem is found. When a child *agent* does not find a solution, it sends a Nogood message to the parent *agent*. The Nogood message contains the variables which empty the variable domains of the child *agent*. When the parent *agent* receives a Nogood message, it stops the search of the children and it tries to find a new solution taking into account the Nogood *information* and so on. Depending on the management of this information, the search space is differently pruned. If a parent *agent* finds a new solution, it will start the same process again sending this new solution to its children. Each *agent* works in the same way with its children in the DFS-tree. However, if the root *agent* does not find solution, then DTH returns *no solution found*.

The heuristic technique is based on Nogood *information*, which allows us to prune search space. First, DTH uses Nogood *information* to prune the search space jumping to the variable involved in the Nogood within the lowest level in the search sequence. Furthermore, DTH does not backtracks in the inverse order of the search, but it jumps directly to other variable involved in the Nogood. Therefore, the search space is pruned in this level of the search sequence and solutions can be deleted. This heuristic technique is not complete. Its level of completeness depend on how Nogood *information* is used.

Figure 5 shows our technique for de-composing a CSP into a DFS-tree CSP structure. Then, DTH is carried out. The root *agent* (a_1) starts the search

process finding a partial solution. Then, it sends this partial solution to its children. The *agents*, which are brothers, are committed to concurrently finding the partial solutions of their subproblem. Each *agent* sends the partial problem solutions to its children *agents*. A problem solution is found when all leaf *agents* find their partial solution. For example, $(\text{state } s_{12} + s_{41}) + (\text{state } s_{12} + s_{23} + s_{31})$ is a problem solution. The concurrence can be seen in Figure 5 in *Time step* 4 in which *agents* a_2 and a_4 are concurrently working. *Agent* a_4 sends a Nogood message to its parent (*agent* a_1) in step 9 because it does not find a partial solution. Then, *agent* a_1 stops the search process of all its children, and it finds a new partial solution which is again sent to its children. Now, *agent* a_4 finds its partial solution, and *agent* a_2 works with its child, *agent* a_3 , to find their partial problem solution. When *agent* a_3 finds its partial solution, a solution of the global problem will be found. It happens in *Time step* 25.

Let's see an example to analyze the behavior of DTH (Figure 6). First the constraint network of Figure 6(1) is partitioned in 3 sub-CSPs and the *DFS tree CSP structure* is built (Figure 6(2)). *Agent* a finds its first partial solution ($X_1 = 1, X_2 = 1$) and sends it to its children: *agent* b and *agent* c (see Figure 6(3)). This is a good partial solution for *agent* c (Figure 6(4)), but this partial solution empties the X_3 variable domain, thus *agent* b sends a Nogood message to its father (Nogood ($X_1 = 1$)) (Figure 6(5)). Then, *agent* a processes the Nogood message, prunes its search space, finds a new partial solution ($X_1 = 2, X_2 = 2$) and sends it to its children (Figure 6(6)). At this point in the process, *agent* c sends a Nogood message to its father (Nogood ($X_1 = 2$)) because X_5 variable domain is empty (Figure 6(7)). *Agent* a stops the search of *agent* b (Figure 6(8)) and then it processes the Nogood message, prunes its search space, finds a new partial solution ($X_1 = 3, X_2 = 3$) and sends it to its children (Figure 6(9)). This last partial solution is good for both children, thus they respond with a OK message and the search finishes (Figure 6(10)).

4.1 DTH: Soundness

If no solution is found, this algorithm terminates. For instance, in Figure 6(10), if *agent* b and *agent* c send Nogood messages, then the root agent empties its domain and terminates with "no solution found".

For the agents to reach a consistent state, all their assigned variable values must satisfy all constraints (*inter-constraints* and *intra-constraints*). Thus, the soundness of DTH is clear.

What remains is that we need to show that DTH must reach one of these conclusions in finite time. The only way that DTH might not reach a conclusion is at least one *agent* is cycling among its possible values in an infinite processing loop. Given DTH, we can prove by induction that this cannot happen as follows.

In the base case, assume that root *agent* is in an infinite loop. Because it is the root agent, it only receives Nogood messages. When it proposes a possible partial state, it either receives a Nogood message back, or else gets no message back. If

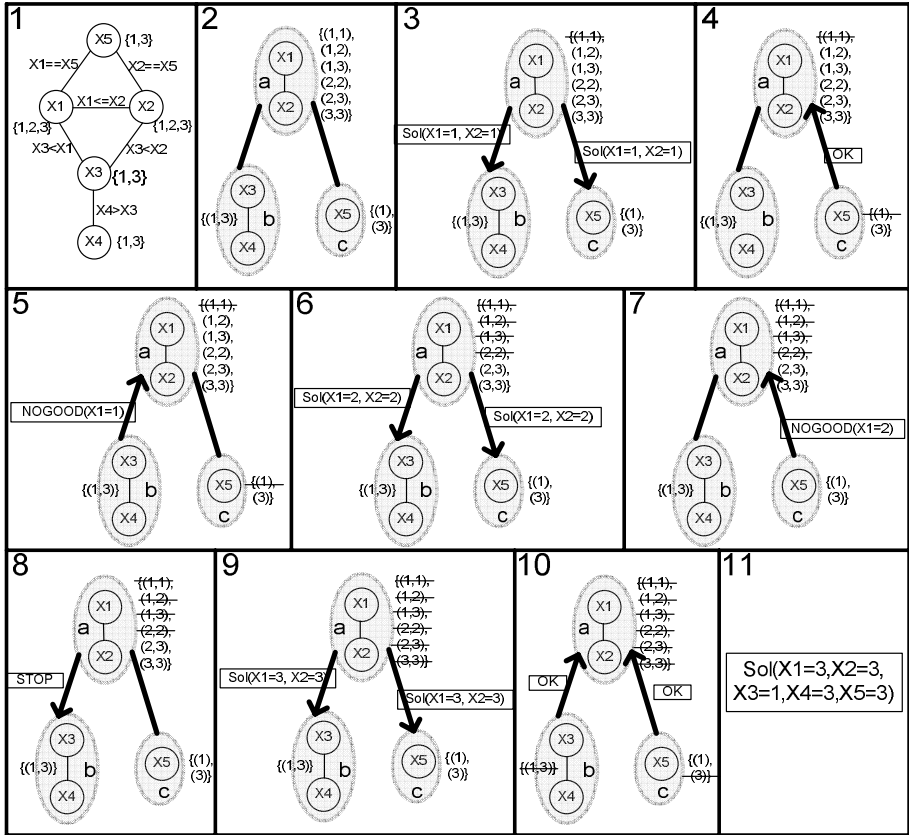


Fig. 6. Example of DTH

it receives Nogood messages for all possible values of its variables, then it will generate an empty domain (any choice leads to a constraint violation) and DTH will terminate. If it does not receive a Nogood message for a proposed partial state, then it will not change that partial state. Either way, it cannot be in an infinite loop. Now, assume that from root *agent* to *agent* in level $k - 1$ in the tree ($k > 2$) maintain a consistent partial state, and *agent* in level k is in an infinite processing loop. In this case, the only messages that *agent* in level k receives are Nogood messages from its children. *Agent* in level k will change instantiation of its variables with different values. Because its variable's domain is finite, this *agent* in level k will exhaust the possible values in a finite number of steps and sends a Nogood message to its parent (which contradicts the assumption that *agent* in level k is in a infinite loop). Thus, by contradiction, *agent* in level k cannot be in an infinite processing loop.

5 Evaluation

In this section, we carry out an evaluation between DTH and a complete CSP solver. To this end, we have used a well-known centralized CSP solver called Forward Checking (FC)².

Experiments were conducted on random distributed networks of binary constraints defined by the 8-tuple $\langle a, n, k, t, p1, p2, q1, q2 \rangle$, where a was the number of sub-CSPs, n was the number of variables on each sub-CSP, k the values in each domain, t was the probability of connection between two sub-CSPs, $p1$ was the *inter-constraint* density for each connection between two sub-CSPs (probability of a non-trivial edge between sub-CSPs), $p2$ was the *intra-constraint* density on each sub-CSP (probability of a non-trivial edge on each sub-CSP), $q1$ was the tightness of *inter-constraints* (probability of a forbidden value pair in an *inter-constraint*) and $q2$ was the tightness of *intra-constraints* (probability of a forbidden value pair in an *intra-constraint*). These parameters are currently used in experimental evaluations of binary Distributed CSP algorithms [3]. The problems were randomly generated by using the generator library in [3] and by modifying these parameters. For each 8-tuple $\langle a, n, k, t, p1, p2, q1, q2 \rangle$, we have tested these algorithms with 50 problem instances of random CSPs. Each problem instance execution has a limited CPU-time (TIME_OUT) of 900 seconds.

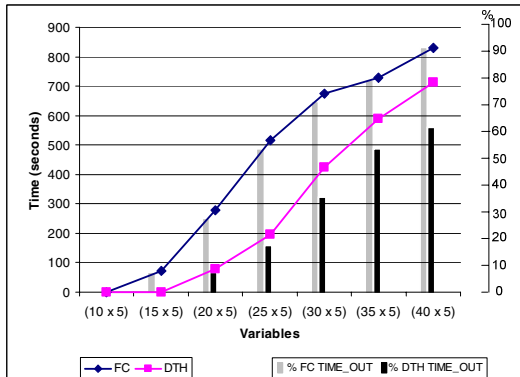


Fig. 7. Running Times and percentage of solution over $TIME_OUT = 900$ seconds with different variable number

In Figures 7 and 8 we compare the running time of DTH with a well-known complete algorithm: Forward Checking. In Figure 7, number of variables on each sub-CSP was increased from 10 to 40, the rest of parameters were fixed ($\langle 5, n, 8, 0.2, 0.01, 0.2, 0.9, 0.3 \rangle$). We can observe that DTH outperformed the FC algorithm in all instances. DTH running times were lower than FC running times. Furthermore, DTH had less TIME_OUT executions than FC. As the number of

² FC was obtained from: <http://ai.uwaterloo.ca/~vanbeek/software/software.html>

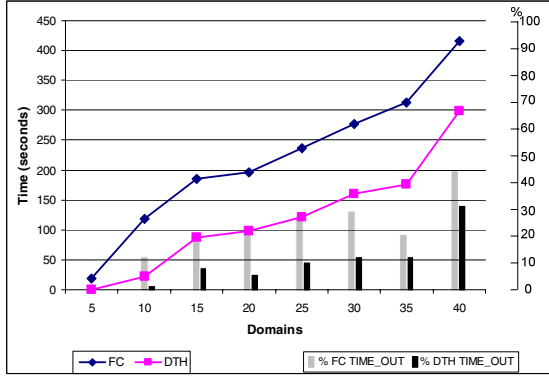


Fig. 8. Running Times and percentage of solution over $TIME_OUT = 900$ seconds with different domain size

variables increased, the number of problem instance executions which achieve $TIME_OUT$ was increased, that is precisely why running time increase more slowly when number of variables increases.

In Figure 8, the domain size was increased from 5 to 40, the rest of parameters were fixed ($< 5, 15, d, 0.2, 0.01, 0.2, 0.9, 0.3 >$). It can be observed that DTH outperformed FC in all cases. As the domain remained greater, the computational cost of DTH and FC also increased. As the domain size increased, the number of problem instance execution which achieve $TIME_OUT$ was increased. Again, DTH had less $TIME_OUT$ executions than FC.

Figure 9 shows the behavior of DTH and FC by increasing q_2 in several instances of q_1 with 5 sub-CSPs, 15 variables and domain size of 10 ($< 5, 15, 10, 0.2, 0.01, 0.2, q_1, q_2 >$). It can be observed that:

- independently of q_1 and q_2 , DTH maintained better behaviors than FC.
- As q_1 increased, problem complexity increased and running time also increased. However, as problem complexity increased, DTH carried out a better pruning. Thus, DTH running times were rather better than FC running times as q_1 increased.
- Problem complexity also depend on q_2 . With regard to q_2 , the most complex problems were those problems generated, in general, with half-values of q_2 ($q_2 = 0.3$ and $q_2 = 0.5$). Moreover, for high values of q_1 , problems were also complex with low values of q_2 ($q_2 = 0.1$). In all these cases, DTH had the best behaviors.

Figure 10 shows the percentages of unsolved problems by DTH when tightness of q_1 increased, that is, problem complexity increased. Problem instances are the same as Figure 9 instances. When problem complexity increased, the number of unsolved solution also increased. However the number of unsolved solution by

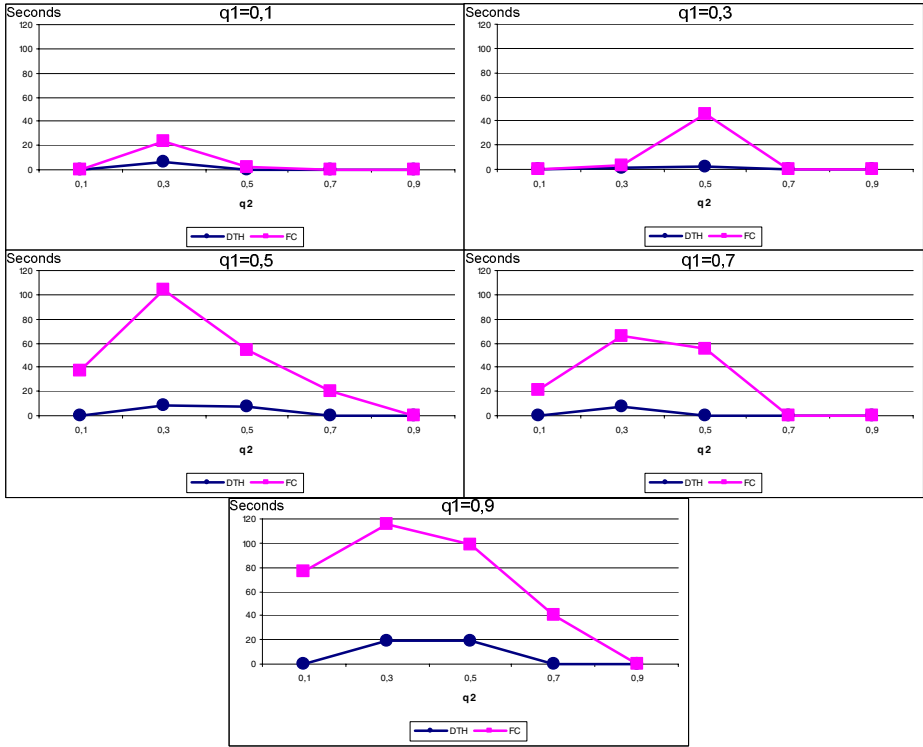


Fig. 9. Running Times with different q_1 and q_2

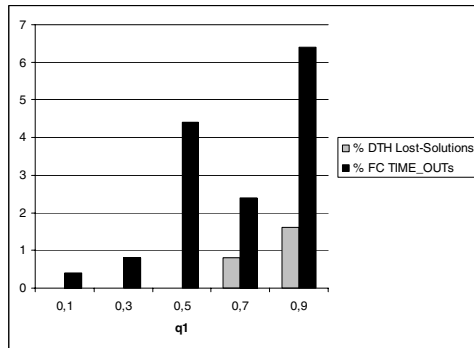


Fig. 10. Percentage of unsolved problems by DTH and percentage of FC TIME_OUTs with different values of q_1

DTH was always smaller than the number of TIME_OUTs when we run FC algorithm. Therefore, our heuristic search is a good option for solving complex decomposable CSPs when we have a time limit.

6 Conclusions

We have proposed an heuristic approach for solving distributed CSP. First, we translate the original CSP into a DFS-tree CSP structure, where each node in the DFS-tree is a sub-CSP. Our heuristic proposal is a distributed heuristic for solving the resultant DFS-tree CSP structure. This heuristic exploits the Nogood *information* for pruning the search space. DTH is sound but not complete. The evaluation shows a good behavior in decomposable CSPs; particularly, the results of the heuristic search are better as problem complexity increases. Furthermore, completeness degree of the heuristics is appropriate for solving decomposable CSPs. Thus, this technique became suitable for solving centralized problems that can be decomposed in smaller subproblems in order to improve solution search.

References

1. Dechter, R.: Constraint networks (survey). *Encyclopedia Artificial Intelligence* , 276–285 (1992)
2. Dechter, R.: *Constraint Processing*. Morgan Kaufmann, San Francisco (2003)
3. Ezzahir, R., Bessiere, C., Belaisaoui, M., Bouyakhf, E.-H.: Dischoco: A platform for distributed constraint programming. In: *Proceedings of IJCAI-2007 Eighth International Workshop on Distributed Constraint Reasoning (DCR'07)*, pp. 16–27 (2007)
4. Hendrickson, B., Leland, R.W.: *A multi-level algorithm for partitioning graphs*. Supercomputing (1995)
5. Karypis, G., Kumar, V.: *Using METIS and parMETIS* (1995)
6. Karypis, G., Kumar, V.: A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of Parallel and Distributed Computing* , 71–95 (1998)
7. Sadeh, N., Fox, M.S.: Variable and value ordering heuristics for activity-based jobshop scheduling. In: *proc. of Fourth International Conference on Expert Systems in Production and Operations Management*, pp. 134–144 (1990)
8. Salido, M.A., Barber, F.: A constraint ordering heuristic for scheduling problems. In: *Proceeding of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications*, vol. 2, pp. 476–490 (2003)
9. Solnon, C.: Ants can solve constraint satisfaction problems. *IEEE Transactions on Evolutionary Computation* 6, 347–357 (2002)
10. Stutzle, T.: Tabu search and iterated local search for constraint satisfaction problems, *Technischer Bericht AIDA9711, FG Intellektik, TU Darmstadt* (1997)
11. Tsang, E.: *Foundation of Constraint Satisfaction*. Academic Press, London and San Diego (1993)
12. Wallace, R., Freuder, E.: Ordering heuristics for arc consistency algorithms. In: *Proc. of Ninth Canad. Conf. on A.I.*, pp. 163–169 (1992)
13. Yokoo, M., Hirayama, K.: Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems* 3, 185–207 (2000)

Partial Pattern Databases

Kenneth Anderson, Robert Holte, and Jonathan Schaeffer

University of Alberta, Computing Science Department,
Edmonton, Alberta, T6G 2E8, Canada
{anderson, holte, jonathan}@cs.ualberta.ca

Abstract. Perimeters and pattern databases are two similar memory-based techniques used in single-agent search problems. We present partial pattern databases, which unify the two approaches into a single memory-based heuristic table. Our approach allows the use of any abstraction level. We achieve a three-fold reduction in the average number of nodes generated on the 13-pancake puzzle and a 27% reduction on the 15-puzzle.

1 Introduction

Perimeters and pattern databases (PDBs) are two successful techniques that improve forward search in single-agent search problems. They have proven effective at improving search performance when combined with minimal-memory, depth-first search techniques such as IDA* [12]. Pattern databases, in particular, have been used to great effect in solving puzzle, DNA sequence alignment, and planning problems [2,18,4].

Perimeters and pattern databases are very similar in their approach to speeding up search. Both techniques use retrograde (reverse) search to fill a memory-based heuristic table. However, pattern databases use abstraction when filling this table, whereas perimeters use none. Also, the memory limit determines the abstraction level for pattern databases; the full PDB must completely fit in memory. Perimeters on the other hand, are built without any abstraction; the perimeter stops being expanded when a memory limit is reached.

We present two general techniques that allow the use of arbitrary abstraction levels in between the two extremes. *Partial pattern databases* use memory similarly to the perimeters, storing part of the space in a hash table. *Compressed partial PDBs* use memory more efficiently, like pattern databases. Our unifying approach allows flexibility when choosing the abstraction level. Through testing, we can determine the best abstraction level for our domains.

We test on two complimentary puzzle domains: the K -pancake puzzle, which has a large branching factor, and the 15-puzzle, which has a small branching factor. Our techniques are compared against full pattern databases, which have proven very effective on these domains. On the 13-pancake puzzle, keeping memory constant, we reduce the average number of generated nodes by a factor of three. On the 15-puzzle, keeping memory constant, we reduce the average number of nodes generated by 27%.

Section 2 examines related work on perimeters, pattern databases, and previous attempts at combining the two approaches. Partial pattern databases and compressed partial PDBs are introduced in Section 3. Results on the K -pancake puzzle and 15-puzzle are shown in Section 4 and Section 5, respectively. Section 6 presents our final analysis and possible extensions to our approach.

2 Background

The domains used in this paper are the K -pancake puzzle and the 15-puzzle. The K -pancake puzzle consists of a stack of K pancakes all of different sizes, numbered 0 to $K - 1$ (Figure 1). There are $K - 1$ operators, where operator k ($1 \leq k \leq K - 1$) reverses the order of the top k pancakes. We refer to an individual pancake as a *tile* and its placement in the stack as a *location*.

The 15-puzzle is comprised of a 4 by 4 grid of tiles, with one location empty. The empty location is called the *blank*. Valid operations include swapping the blank with one of up to 4 adjacent tiles. Figure 1 shows a possible arrangement of tiles for the 15-puzzle.

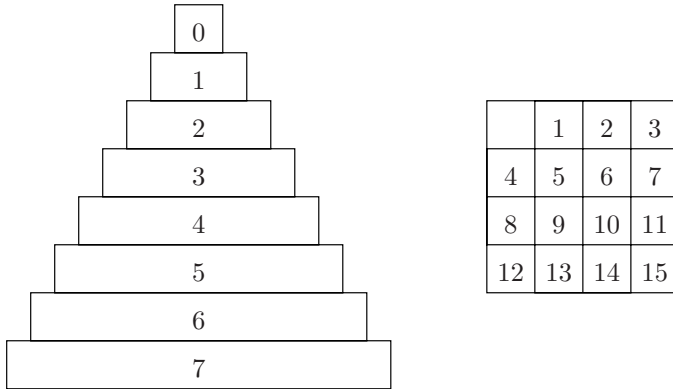


Fig. 1. Goal node for 8-pancake (left) and 15-puzzle (right)

A *node* is a unique arrangement of tiles. Given a *start* and *goal* node, we wish to find a path (series of operators) of minimal cost leading to the goal node. In our domains, operators have unit cost. However, the ideas in this paper are applicable to any domain that can be structured as a graph with directed, weighted edges of non-negative cost.

IDA* is a traditional search technique proven to work well in these two domains. IDA* is a depth-first search method that finds optimal solution paths through a directed weighted graph [12]. Nodes are pruned if the total cost through a node, $f(n)$, exceeds a bound f_{limit} according to the definition $f(n) = g(n) + h(n)$, where $g(n)$ is the actual cost to node n and h is the heuristic estimate of the remaining cost to the goal. IDA* iteratively increases the bound, f_{limit} ,

and re-searches until the goal is found. If h is *admissible*, does not overestimate the cost to the goal, then IDA* guarantees finding an optimal path, provided that one exists.

If a heuristic is *consistent*, then the change in heuristic between any two nodes in the graph is less than or equal to the cost between the two nodes. Consistency implies admissibility. If IDA* is used with an inconsistent heuristic, heuristic values can be improved through the use of bidirectional-pathmax (BPMX) [8]. BPMX propagates heuristic inconsistencies through the search tree from parent-to-child and child-to-parent, increasing heuristic values when appropriate. This technique uses no extra memory in IDA* and takes very little time.

The *original space*, S , consists of the set of nodes that can reach the goal through a series of operators. An *abstract space* is a set of abstract nodes and all applicable operators, where every node in the original space maps to some corresponding node in the abstract space. We use *domain abstraction* as our abstraction technique [10]. The abstraction is created by renaming specific tiles to the same name, x . These re-named tiles are called *don't-care* tiles while the other tiles are called *unique tiles*. In the case of the 15-puzzle, the blank will always be a unique tile. When we refer to *abstraction- N* , we refer to a specific abstraction with N unique tiles (the actual tiles vary with the domain). A *coarse-grained* abstraction has fewer unique tiles (and hence covers more abstract nodes) than a *fine-grained* abstraction.

The following are memory-based heuristic methods. Both techniques use *retrograde* (backwards) search from the goal to improve or create heuristic values, but they achieve this in quite different ways.

2.1 Perimeters

Perimeter search is a type of bidirectional search technique and requires a predecessor function. Originally proposed by Dillenburg and Nelson, perimeter search performs two successive searches [3]. The first search proceeds in the backwards direction from the goal, forming a set of perimeter nodes P , which encompass the goal. A node n is *on* the perimeter if $n \in P$, *inside* the perimeter if it was expanded during perimeter creation, and *outside* the perimeter if it was not expanded. Any node outside the perimeter must pass through some node in the perimeter to reach the goal. Many techniques can be applied to generate the perimeter: Breadth-first search creates a *constant-depth perimeter* [3,14]; A* creates a *constant-evaluation perimeter* [3]; and expansion based on heuristic difference creates another kind of perimeter [11].

If the perimeter is generated for one problem instance, then the backward and forward searches are performed in series. The backward search forms a perimeter and, if the start node is not found, it is followed by the forward search. However, if the perimeter is constructed for use on multiple problem instances with the same goal, then the interior of the perimeter, set A , is stored. When the forward search begins, if the start is in set A , the actual cost to the goal is known so the heuristic is corrected to this value. Otherwise the start is outside the perimeter and the forward search begins.

The second, forward search progresses either from the start to the perimeter, called *front-to-front* evaluation, or from the start to the goal, called *front-to-goal* evaluation. Front-to-front evaluation calculates the heuristic value of a node based on the estimated cost through every node on the perimeter. Although larger perimeters provide better heuristic values, the heuristic takes increasingly longer to compute. Additionally, front-to-front evaluation requires a heuristic to exist between any two distinct nodes.

By contrast, search using front-to-goal evaluation requires only a heuristic to the goal. The heuristic of nodes found to be inside the perimeter is corrected using the exact cost to the goal. The heuristic of nodes outside the perimeter can sometimes be corrected; here are two different approaches for correcting the heuristic values using front-to-goal evaluation. Using a depth-limited perimeter where d is the cost bound, d is a lower bound on the true cost to the goal for all nodes outside the perimeter [1]. Or, given a consistent heuristic, a correction factor is equal to the lowest difference between the actual cost to the goal and the estimated heuristic cost to the goal [11]. The correction factor is now added to the original heuristic if outside the perimeter.

Any search technique will work for the forward search, but IDA* and similar low-memory search techniques are most commonly employed for their adaptability, scalability, and low memory requirements [3,14,11]. We use IDA* throughout this paper.

2.2 Pattern Databases

Introduced by Culberson and Schaeffer, pattern databases use abstraction to create a heuristic lookup table [2]. Retrograde search starts from the abstract goal. The search proceeds backwards applying all applicable reverse operators until the space is covered. The costs from the abstract goal in the abstract space are recorded in a table and used as a heuristic in the forward search. This produces an admissible and consistent heuristic.

Holte *et al.* have investigated generating and caching parts of pattern databases during search in [9]. Similarly, Zhou and Hansen have demonstrated a technique whereby provably unnecessary parts of the PDB are not generated (given an initial consistent heuristic and an upper bound on solution length) [17]. Felner and Adler further iterated upon on this procedure using *instance dependent pattern databases* [5]. We approach this problem from the opposite position; how can we create and use part of a pattern databases and/or a perimeter over multiple problem instances?

2.3 Perimeter and PDB Comparison

Perimeters and pattern databases are similar in many respects. Both perimeters and pattern databases require a predecessor function. This predecessor function enables retrograde search from the goal. Both procedures can also be improved by using domain-specific properties: perimeters by using a heuristic function

between two arbitrary nodes, and pattern databases through symmetry [2], additivity [6], and duality [8,16].

The two techniques also differ in critical ways. First, pattern databases require a node abstraction mechanism, which perimeters avoid. This freedom allows perimeters to be applied to domains without any known abstraction. On the other hand, where perimeter search requires an available heuristic between any two nodes, pattern databases can generate a heuristic for domains where one is not known. Finally, PDBs cover the entire space, while perimeters only cover part of it (Figure 2). As a result, each node in the perimeter must store a node identifier as well as the cost to the goal. In general, any partial set of the original or abstract space requires extra memory to store the node identifier information. Perimeters fall into this category, as do instance-dependent pattern databases. On the other hand, because pattern databases cover the entire domain space, heuristic values may be indexed by their *nodeID* (the node need not be stored for each entry (Figure 3)).

2.4 Combining Perimeters with PDBs

Perimeter search works well for correcting pre-existing heuristics [3,14,11], while pattern databases prove valuable for creating a heuristic to the goal node [2]. In the remainder of this paper, we propose and investigate a new, general method for combining these two techniques into a single lookup table.

Culberson and Schaeffer use a pattern database as a heuristic simultaneously with a perimeter [1]. Because the pattern database only provides a heuristic to the goal node, perimeter search must use a front-to-goal evaluation technique. If a node is in the perimeter, we know the actual cost to the goal and use that for the heuristic value. A perimeter with cost-bound d has the following property: d equals the minimum cost to the goal of all nodes outside the perimeter. This means that for any node n not inside this perimeter, $h(n) \geq d$ and is corrected accordingly. In fact, as long as d is defined as above, this can be applied to any shaped perimeter.

In a concurrent submission to this SARA symposium, Ariel Felner uses a perimeter to *seed* a pattern database [7]. The pattern database is built using the perimeter as the goal node. This represents an alternative procedure for combining the techniques of perimeter search with pattern databases, but differs significantly from the approach that we present in Section 3.

For every node on the perimeter, Kaindl and Kainz track the difference between the actual cost to the goal and the heuristic value [11]. We call the minimal difference value for all nodes on the perimeter the *heuristic correction factor*. The perimeter is built by expanding the node with the smallest difference, to increase the heuristic correction factor. If the heuristic is consistent, they admissibly add the heuristic correction factor to every node outside the perimeter. A pattern database is a consistent heuristic, so this technique is applicable. However, Felner has shown the following is true for the 15 puzzle using our abstraction method:

given an abstract node a with an abstract cost c to the abstract goal, there exists a node in the original space mapping to a with a cost c from the goal [7]. The same is true for the K -pancake puzzle. Consider using a perimeter built by expanding nodes with the lowest heuristic value [11]. To obtain a correctional factor equal to one, the perimeter must have at least as many entries as the pattern space. Because of the additional memory required by perimeters to store the *nodeID* (see Figure 3), this method is impractical for our domains.

3 Partial Pattern Databases

Research in pattern databases is beginning to incorporate approaches typically seen in perimeter search. Specifically, subsets of full pattern databases are being stored in the form of *instance-specific* PDBs and caching in hierarchical search [17, 5, 9]. We take this one step further by storing part of a full PDB. Our approach is not instance-specific; it reuses the same database over multiple search instances with a fixed goal.

A *partial pattern database* consists of a set of abstract nodes A and their cost to the goal, where A contains all nodes in S with cost to goal less than d . d is a lower bound on the cost of any abstract node not contained in A . In essence, a partial pattern database is a perimeter in the abstract space (with the interior nodes stored). Any node n in the original space has a heuristic estimate to the goal: if n is in the partial PDB, return the recorded cost; if n is not in the partial PDB, return d . This heuristic is both admissible and consistent.

Building a partial PDB is similar to building a perimeter, only in the abstract space. Retrograde search is performed starting from the abstract goal. The heuristic values are recorded. When a memory limit is reached, the partial PDB building stops and heuristic values are used for the forward search. Depth d is the minimum cost of all abstract nodes outside the partial PDB. Note that all abstract nodes in A with cost equal to d can be removed from the partial PDB to save memory. This will not affect the heuristic.

On one extreme, a partial PDB with no abstraction reverts to exactly a perimeter (with the interior nodes stored). On the other extreme, a partial PDB with a coarse-grained abstraction will cover the entire space, and performs exactly like a full PDB. However, a partial PDB does not store the data as efficiently as a full PDB.

3.1 Memory Requirements

A full PDB encompasses the entire space (Figure 2); every node visited during the forward search has a corresponding heuristic value in the lookup table. The PDB abstraction level is determined by the amount of available memory. Finer-grained abstraction levels are not possible because the memory requirements increase exponentially with finer abstractions.

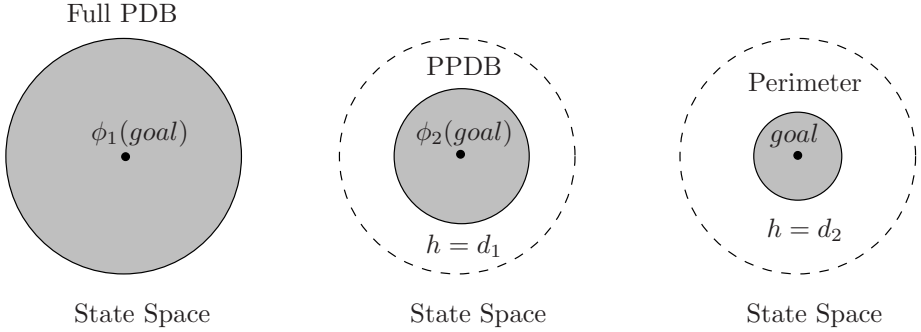


Fig. 2. Coverage of original space by lookup tables (full PDB on left, partial PDB in middle, and perimeter on right)

Partial PDBs, as perimeters, generally do not cover the entire space. During the forward search, if a node is not contained in the partial PDB lookup table, d is returned. Partial PDBs add flexibility over full PDBs by allowing the use of any abstraction level. However, there are drawbacks with respect to memory requirements.

Because full pattern databases cover the entire domain space, the heuristic values in the lookup table are indexed by their unique node identifier ($nodeID$). Therefore, a table of the exact size of the abstract space can be used and there is no need to store the $nodeID$ (Figure 3(a)). Memory is only used to store the abstract cost to the goal. On the other hand, perimeters, instance-dependent PDBs [17,5], and partial PDBs only cover part the space. As a result, each node in the perimeter must store the $nodeID$ in addition to the abstract cost to the goal (Figure 3(b)). This requires extra memory for every table entry.

In our domains, the 15-puzzle and the pancake puzzle, partial pattern database entries require nine times more memory than full pattern database entries. This is a severe limitation on the effective use of partial pattern databases.

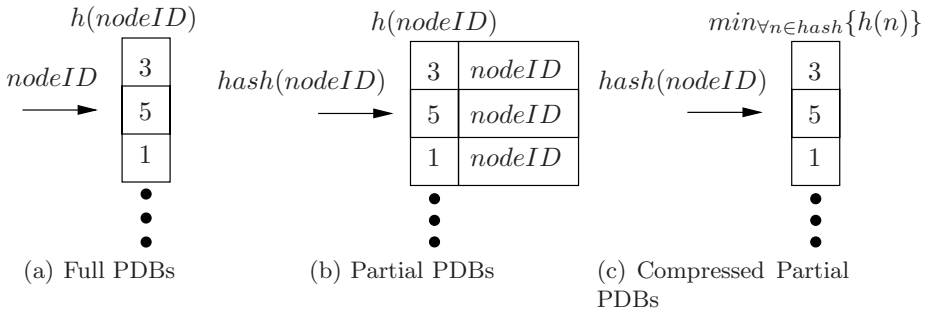


Fig. 3. PDB storage strategies

3.2 Compressed Partial PDBs

For perimeters and partial pattern databases, the added cost of storing a node’s identification information is an expensive use of memory. To maintain efficiency, the hash table should have a reasonable fill-factor, but space is inevitably wasted on empty table positions. We therefore present a compressed version of a partial PDB that does not store the *nodeID* and can be filled to any convenient fill-factor.

Given an abstraction granularity, our hash function maps each abstract node to a location. For all abstract nodes mapped to the same location, we store the minimum heuristic value (to preserve admissibility) (Figure 3(c)). When we query for a node’s heuristic value, we return the value stored in the table at that node’s hashed location. This heuristic value is guaranteed to be admissible, but it may be inconsistent.

The creation of compressed partial PDBs occurs as a preprocessing step. Therefore, we can take a large amount of time, use machines with more available memory, or use disk-based algorithms. The simplest technique is to use iterative-deepening depth-first search from the abstract goal, filling a value in the table if it is lower than the existing entry. However, unlike with full PDBs, if a node’s heuristic value is larger than the entry in the table, it cannot be cut-off without breaking admissibility. Therefore, this *depth-first construction* method must use a complimentary technique to remove transpositions; we use a transposition table [15]. This is the technique used in Section 4.

A second technique is to use breadth-first search. This removes transpositions, but uses a large amount of memory for construction. However, delayed duplicate-detection is an efficient disk-based algorithm that can be used to do this [13]. In the interest of simplicity, we did not attempt this technique.

A third alternative is to build a full PDB at a fine-grained level using a machine with a large amount of memory. Full pattern databases can be computed very efficiently using iterative-deepening depth-first search with only a small amount of excess memory. This is the approach used in Section 5. The main drawback to this approach is that it does not scale well to very fine-grained abstractions. However, our results in Section 4 show that very fine-grained abstractions are not necessary.

One item worth consideration is the hashing function. As the authors found out, a simple modular hashing scheme can introduce regularity in the table. In the worst case, a fine-grained compressed partial PDB can revert to exactly the coarser-grained version. Also, depending on the hash function, the table may not fill to 100%.

4 Experiments on the K-Pancake Puzzle

The abstractions used for the *K*-pancake puzzle have don’t-care tiles as the low-indexed tiles. For example, abstraction-7 for the 12-pancake puzzle refers to the abstraction ‘*x x x x x 5 6 7 8 9 10 11*,’ with *x* as a don’t-care. Note that abstraction-11 is the finest-grained abstraction and is the same as the original

space because there are 12 distinct tiles. We use the depth-first construction technique to create the compressed PDBs.

4.1 Performance with Constant a Number of Entries

Throughout this section, we report the average number of nodes generated during the forward search as our metric. Each data point is an average of 100 random starting nodes. Partial pattern databases of various abstraction levels are used for the heuristic. The number of entries in the partial PDB remains constant (specified in the results tables), while the level of abstraction varies. IDA* with BPMX is used for the forward search. Recall that BPMX only has an effect if the heuristic is inconsistent; therefore, only the compressed partial pattern database will be affected by using BPMX. Also, note that this is only a comparison between partial pattern databases; full pattern databases make much more efficient use of memory. We will cross-compare techniques in Section 4.2.

Table 1 shows the average number of nodes generated on the 8, 10, 12, and 13-pancake puzzles. Each puzzle fixes the number of entries in the partial PDB to exactly the number of entries of a full PDB with $\lfloor K/2 \rfloor$ unique tiles. For instance, the 12-pancake puzzle partial PDBs each have 665,280 entries (six unique tiles). The top data point of each column generates the same number of nodes as a full pattern database, while the bottom data point generates the same number of nodes as a perimeter.

Table 1. Average number of nodes generated using a single Partial PDB on K -Pancake puzzle

Number of Unique Tiles	8-Pancake 1,680 entries (cost-bound d)	10-Pancake 30,240 entries (cost-bound d)	12-Pancake 665,280 entries (cost-bound d)	13-Pancake 1,235,520 entries (cost-bound d)
4	2,065 (7)			
5	682 (5)	48,408 (9)		
6	403 (5)	14,268 (6)	1,316,273 (11)	25,833,998 (12)
7	335 (5)	8,251 (6)	178,464 (8)	3,106,345 (8)
8		7,370 (6)	183,172 (7)	4,097,683 (7)
9		7,242 (6)	167,584 (7)	3,851,260 (7)
10			165,390 (7)	3,820,667 (7)
11			164,951 (7)	3,816,931 (7)
12				3,819,909 (7)

We see that with the same number of entries, but using a finer granularity partial PDB, the average number of nodes generated reduces. Note however, that in the 12-pancake puzzle, abstraction-7 produced fewer nodes than abstraction-8. Search performance is very sensitive to the cost-bound of the partial pattern database. The increase in nodes generated from abstraction-7 to abstraction-8 occurs because abstraction-7 has a larger cost-bound d than abstraction-8.

Consider two partial pattern databases with the same cost-bound d , but one being based on a coarser granularity than the other. The finer-grained database will *dominate* the coarser-grained one because for every node n in the space, $h_{fine}(n) \geq h_{coarse}(n)$. Abstraction-8, 9, 10, and 11 on the 12-pancake puzzle demonstrate this principle.

We will continue to see this trade-off between abstraction granularity and database cost-bound throughout the results. We can think of this intuitively as follows: making a partial PDB finer-grained improves the heuristic value of nodes inside the database at the cost of nodes outside the database (if d gets smaller). At some point the heuristic outside the partial PDB becomes so inaccurate that the performance is dominated by these nodes. Refining the abstraction further from this point generally produces worse performance. Analogous results are documented in [10], where increasing small heuristic values improves performance, but only up to a point.

Table 2. Average number of nodes generated using a single compressed partial PDB filled to 70% on K -Pancake puzzle

Number of Unique Tiles	8-Pancake 1,680 entries (cost-bound d)	10-Pancake 30,240 entries (cost-bound d)	12-Pancake 665,280 entries (cost-bound d)	13-Pancake 1,235,520 entries (cost-bound d)
4	2,065 (7)			
5	1,024 (6)	48,414 (8)		
6	1,227 (6)	18,026 (8)	1,316,284 (10)	25,834,132 (10)
7	1,564 (6)	22,117 (7)	358,585 (9)	6,481,829 (9)
8		29,867 (7)	379,655 (9)	6,599,913 (9)
9		39,080 (7)	520,648 (8)	10,142,002 (9)
10			677,805 (8)	15,214,336 (8)
11			841,576 (8)	22,068,332 (8)
12				27,498,382 (8)

Table 2 shows the average number of nodes generated over 100 search instances using compressed partial PDBs. The compressed database is filled to 70% full. We examine the 8, 10, and 12-pancake puzzles.

The top entry of each column matches closely with the performance of a full pattern database (shown at the top of the corresponding columns in Table 1). At this abstraction level, each entry corresponds to one abstract node; if the compressed partial PDB were filled to 100% and had no hash collisions, then it would be identical to a full PDB. However, these conditions do not hold in general, so the top entries in each table do not match exactly.

Let us examine another two corresponding entries in each table, 12-Pancake at abstraction-7. For the partial pattern database (Table 1), there is an average of 178,464 nodes generated. For the compressed partial PDB (Table 2), there are 358,585 generated nodes on average. Keep in mind that both tables have the same number of entries, but the entries themselves are different.

The compressed partial PDB generates more nodes for three reasons. First, the table is only filled to 70% full, but this has a small effect because the unreached entries are filled with d , limiting the heuristic error. Second, the heuristic values in the perimeter are degraded by taking the minimal value of all nodes hashed to the same location. Third and most importantly, the heuristic correction of d is not applied to all nodes outside the perimeter. Because we store the minimum heuristic value, nodes outside the perimeter overlap with nodes inside the perimeter, reducing the effect of the heuristic correction factor.

However, because of the construction method, the partial PDB does not necessarily dominate the compressed partial PDB. The compressed partial PDB is filled until 70% full. Overlapping nodes cause the table to fill more slowly than the partial PDB. Thus the final cost-bound d may be greater in the compressed partial PDB than the partial PDB. This is seen in our example with the 12-Pancake puzzle at abstraction-7: the partial PDB is built to $d = 8$ while the compressed partial PDB is built to $d = 9$.

As the granularity becomes finer, we quickly reach the point of diminishing returns. In all examples, this occurs after adding one more unique tile to the original PDB abstraction. For the 8, 10, 12, and 13-pancakes, the optimal granularity is 5, 6, 7, and 8 unique tiles respectively. Further refining of the abstraction only causes the number of generated nodes to increase. This is caused by the poor, high-valued heuristics.

The *improvement factor* is the improvement over the original abstraction (top entry in Table [II](#)). This factor increases with puzzle size. The 8, 10, 12, and 13-pancake puzzles' best performance factors are 50%, 63%, 73%, and 75%. This indicates that savings may scale favorably to larger problems.

4.2 Performance with Constant Memory

As stated, partial pattern databases need extra memory to store the *nodeID*. In the case of the 10-pancake puzzle, the amount of memory used to store the *nodeID* is about eight times larger than the heuristic value that is stored. To directly compare partial PDBs with full PDBs and compressed partial PDBs, we need to keep memory constant. So we limit the number of entries in the partial PDB appropriately.

Each full PDB entry consists of one byte, as does each compressed partial PDB entry. Each partial PDB entry consists of 9 bytes (1 for the heuristic value and 8 for the node id) plus extra room in the hash table for empty entries (fill factor). We therefore calculate an approximate number of entries for the partial PDB that fits into the designated number of bytes using the formula: $entries_{partialPDB} = bytes/9 * 0.7$.

Table [3](#) directly compares the performance of the three heuristic techniques (full PDBs, partial PDBs, and compressed partial PDBs) on the 10-pancake puzzle while keeping memory constant. Each data entry is the average number of generated nodes over 100 random instances. For the partial PDB and compressed partial PDB, we tested from one to nine unique tiles; this table shows only the best result. The associated number in parentheses depicts the number of unique

Table 3. Average number of nodes generated using a single PDB of the best abstraction granularity. The tests are performed on the 10-Pancake puzzle while keeping memory constant.

10-Pancake Puzzle			
Memory Limit	Normal PDB (unique tiles)	Best Partial PDB (unique tiles)	Best Compressed Partial PDB (unique tiles)
3,628,800 bytes	40 (9)	2,003 (7)	40 (9)
1,814,400 bytes	200 (8)	4,628 (7)	70 (9)
604,800 bytes	1,216 (7)	16,147 (6)	417 (8)
151,200 bytes	7,756 (6)	78,073 (5)	2,695 (7)
30,240 bytes	48,408 (5)	511,794 (5)	18,026 (6)
5,040 bytes	337,021 (4)	3,299,716 (4)	135,352 (5)

tiles used to generate the data point. For each memory limit, we show the best-performing database in bold.

Partial PDBs by themselves are not an efficient use of memory; as in all tested cases, partial PDBs generate at least an order of magnitude more nodes than full PDBs. However, the compressed partial PDBs are an efficient use of memory. The top row covers the entire space at the finest granularity; this is a perfect heuristic. In this case the full PDB has slightly better performance because the compressed partial PDB is only filled to 70% (this is not apparent in the table because of averaging). The improvement factor for every row except the first is between 60% and 65%, indicating similar performance gains when the memory limit is smaller than the size of the abstract space.

5 Experiments on the 15-Puzzle

The abstractions used for the 15-puzzle are as follows: abstraction-8 is the *fringe* [2], ‘b x x 3 x x x 7 x x x 11 12 13 14 15’; and abstraction-9 adds one more unique tile, ‘b x x 3 x x x 7 x x 10 11 12 13 14 15’. The heuristic used is the maximum of Manhattan Distance and the PDB or compressed partial PDB lookup. No symmetries or other search enhancements are used.

Each test is run over all 100 Korf problem instances [12]. The databases compared are the full pattern database with abstraction-8 (PDB_8) and the compressed partial pattern database using abstraction-9 ($PPDB_9$). $PPDB_9$ is created from the full PDB using abstraction-9. We use IDA* with bidirectional pathmax (BPMX) to take advantage of the inconsistency in the compressed partial PDBs.

The columns of Table 4 are as follows:

- The *PDB* is the type of pattern database used: either the full pattern database PDB_8 or the compressed partial pattern database $PPDB_9$.
- *Memory* is the number of bytes in the database: in this case held constant at 518,918,400 bytes.

Table 4. Nodes generated on the 15-puzzle using a single PDB technique and Manhattan Distance while keeping memory constant

PDB	Memory	BPMX	Fill	d	small	medium	large	total
PDB_8	518,918,400	DC	100	64	283,309	6,929,803	80,291,808	1,067,439,170
$PPDB_9$	518,918,400	Yes	50	38	792,425	14,465,508	123,546,247	1,840,334,929
$PPDB_9$	518,918,400	Yes	60	39	651,152	12,181,920	101,463,007	1,525,898,811
$PPDB_9$	518,918,400	Yes	70	40	554,045	10,529,666	86,096,055	1,304,112,453
$PPDB_9$	518,918,400	Yes	80	41	487,551	9,361,520	75,459,248	1,149,450,912
$PPDB_9$	518,918,400	Yes	90	43	409,084	7,943,644	62,933,272	965,285,000
$PPDB_9$	518,918,400	Yes	98	66	330,510	6,473,794	50,611,367	780,456,393
$PPDB_9$	518,918,400	No	98	66	539,065	9,917,842	82,560,198	1,242,278,013

- *BPMX* tells whether bidirectional pathmax is used: *Yes* it is used, *No* it is not used, and *DC* (don't care) means that *BPMX* has no effect.
- *Fill* shows the percentage of memory that is expanded when creating the pattern database. Because of the hashing scheme, however, even when filled completely, 2% of $PPDB_9$ remains unexpanded.
- For PDB_8 , d is the largest value in the database. For $PPDB_9$, d is the cost bound (as defined in Section 3).
- The *small* problems are the problems that result in searches with less than 1,000,000 generated nodes when solving with PDB_8 . *Medium* problems are between 1,000,000 and 31,999,999 generated nodes. The *hard* problems are greater than or equal to 32,000,000 generated nodes. There are 43 small problems, 48 medium problems, and 9 hard problems. We report the average number of nodes expanded in each of the three problem sets.
- *total* is the sum of all generated nodes over the 100 Korf problem instances.

At less than 90% full, using *BPMX*, $PPDB_9$ generates more total nodes than PDB_8 . $PPDB_9$ generates slightly fewer nodes when at 90% full, and at 98% full the total number of generated nodes is decreased by 27%. However, this result can be slightly misleading, since the total is dominated by the largest searches (which build over three orders of magnitude larger search trees). Thus, we also examine the problems grouped by difficulty. With and without *BPMX*, the small problems have worse performance using $PPDB_9$ than PDB_8 . However, the large problems have improved performance when using *BPMX* and filled to 80% or more. When $PPDB_9$ is 98% full, the small, medium, and large problems have -95%, 6%, and 36% improvement respectively, in average number of nodes generated. Not only does the total number of generated nodes decrease by 27%, but the hard instances are improved the most.

The last row shows the importance of using *BPMX* in the 15-puzzle to improve performance. Using *BPMX* leads to a 37% reduction in the total number of nodes generated when $PPDB_9$ is 98% full. This pushes the performance ahead

of PDB_8 . Small, medium, and large instances benefit equally from BPMX, improving the number of generated nodes by 39%, 35%, and 39% respectively. This indicates that the influence of BPMX may not depend on instance difficulty.

6 Conclusions

This paper presents partial pattern databases, a general approach that merges the ideas of front-to-goal perimeter search and full pattern databases. Our approach decouples the abstraction granularity from the database size, freeing the programmer to use the best abstraction for a given domain and amount of memory. Partial PDBs are applicable to domains with a predecessor function and a space abstraction technique and can be re-used over multiple problem instances with the same goal.

Two versions are presented: the original partial pattern databases, which store the heuristic value and the *nodeID*; and the more memory-efficient compressed partial PDBs, which store only one heuristic value. Two complementary puzzle domains are tested: the K -pancake puzzle, which has a large branching factor of $K - 2$, and the 15-puzzle, which has an average branching factor of 2.1.

Compressed partial pattern databases are shown to be most effective on the K -pancake puzzle; the number of nodes generated on the 13-pancake puzzle is reduced by three-fold. Uncompressed partial pattern databases are not shown to be effective for this domain because of the memory inefficiency.

On the 15-puzzle, compressed partial PDBs in combination with the Manhattan Distance heuristic are slightly less successful. This is because MD often corrects a PDB's low heuristic values, which is one of the primary benefits of using partial PDBs. However, in combination with BPMX, and filled to 98%, we are able to reduce total number of nodes generated by 27%. Interestingly, hard problem instances are better improved than small instances.

This paper presents, implements, and tests partial PDBs in general terms. This technique can be further incorporated with other general methods. For example, the maximum can be taken over multiple heuristic lookup tables, whether they be PDBs, partial PDBs, or compressed partial PDBs [10]. As well, domain-specific adaptations and improvements can be integrated into this framework. Two glaring examples are additivity in the 15-puzzle and duality in the pancake puzzle. On the 15-puzzle, partial PDBs can always be made into additive versions by ignoring don't-care tile movements. Compressed partial PDBs can effectively compress larger full additive pattern databases into memory-efficient versions. On the pancake puzzle, we can use any pattern database technique to get a heuristic to the goal. By using the *general duality principal* [8], we can get an admissible heuristic between any two nodes (map the operator sequence II between two nodes onto the goal to get node n^d , and look up $h(n^d)$ from the PDB). This would allow for front-to-front heuristic improvement using a partial pattern database, effectively coming full-circle back to the original perimeter search technique.

Acknowledgments

We would like to express our thanks and appreciation to the reviewers for their comments and corrections, Cameron Bruce Fraser and David Thue for proof-reading, and Ariel Felner for his thought-provoking discussions. This work was supported by NSERC and iCORE.

References

1. Culberson, J.C., Schaeffer, J.: Efficiently searching the 15-puzzle. Technical Report TR 94-08, Department of Computing Science, University of Alberta (1994)
2. Culberson, J.C., Schaeffer, J.: Searching with pattern databases. In: Canadian Conference on AI, pp. 402–416 (1996)
3. Dillenburg, J.F., Nelson, P.C.: Perimeter search. *Artificial Intelligence* 65(1), 165–178 (1994)
4. Edelkamp, S.: Planning with pattern databases. In: ECP: European Conference on Planning, Toledo, pp. 13–34 (2001)
5. Felner, A., Adler, A.: Solving the 24 puzzle with instance dependent pattern databases. In: Zucker, J.-D., Saitta, L. (eds.) SARA 2005. LNCS (LNAI), vol. 3607, pp. 248–260. Springer, Heidelberg (2005)
6. Felner, A., Korf, R.E., Hanan, S.: Additive pattern database heuristics. *JAIR: Journal of Artificial Intelligence Research* 22, 279–318 (2004)
7. Ariel Felner and Nir Ofek. Combining perimeter search and pattern database abstractions. In: Symposium on Abstraction Reformulation and Approximation (SARA) (2007)
8. Felner, A., Zahavi, U., Schaeffer, J., Holte, R.: Dual lookups in pattern databases. In: IJCAI, pp. 103–108 (2005)
9. Holte, R.C., Grajkowski, J., Tanner, B.: Hierarchical heuristic search revisited. In: Zucker, J.-D., Saitta, L. (eds.) SARA 2005. LNCS (LNAI), vol. 3607, pp. 121–133. Springer, Heidelberg (2005)
10. Holte, R.C., Newton, J., Felner, A., Meshulam, R., Furcy, D.: Multiple pattern databases. In: ICAPS, pp. 122–131 (2004)
11. Kaindl, H., Kainz, G.: Bidirectional heuristic search reconsidered. *Journal of Artificial Intelligence Research* 7, 283–317 (1997)
12. Korf, R.E.: Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence* 27(1), 97–109 (1985)
13. Korf, R.E.: Delayed duplicate detection: Extended abstract. In: Gottlob, G., Walsh, T. (eds.) IJCAI, pp. 1539–1541. Morgan Kaufmann, San Francisco (2003)
14. Manzini, G.: BIDA: An improved perimeter search algorithm. *Artificial Intelligence* 75(2), 347–360 (1995)
15. Reinefeld, A., Marsland, T.A.: Enhanced iterative-deepening search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16(7), 701–710 (1994)
16. Zahavi, U., Felner, A., Holte, R., Schaeffer, J.: Dual search in permutation state spaces. In: AAI, pp. 1076–1081 (2006)
17. Zhou, R., Hansen, E.A.: Space-efficient memory-based heuristics. In: AAI, pp. 677–682 (2004)
18. Zhou, R., Hansen, E.A.: External-memory pattern databases using structured duplicate detection. In: AAI, pp. 1398–1405 (2005)

CDB-PV: A Constraint Database-Based Program Verifier*

Scot Anderson¹ and Peter Revesz²

¹ Southern Adventist University, Collegedale, TN 37315, USA
scot@southern.edu

² University of Nebraska-Lincoln, NE 68588, USA
revesz@cse.unl.edu

Abstract. In this paper we present a new system called CDB-PV that uses constraint databases (CDBs) for program verification (PV). The CDB-PV system was implemented in C++ and tested on several sample programs that are difficult to verify using other methods. The CDB-PV system also runs efficiently for the sample programs. The CDB-PV approach is similar to abstract interpretation but it allows non-convex approximations.

1 Introduction

Programs increasingly control many aspects of our daily lives such as air traffic control (ATC) systems. Failures such as the computer controlled Airbus A320 crash on June 26, 1988 [1] show that programs need thorough debugging and verification before risking lives. We propose a new constraint database approach to program debugging and verification.

Verifying the correctness of programs is undecidable in general. That is easy to see by looking at the well-known halting problem, which is the problem of deciding whether a given program with a given input will terminate. Since the halting problem is undecidable in general and termination of programs is usually considered one of the conditions of correctness, it is clear that program verification is also undecidable in general.

However, let us take a deeper look at program verification and identify what can be done. We start with the following definitions.

Definition 1 (Program State). *A program state is a pair consisting of the values assigned to the program variables and the specific location of the program code where such an assignment occurs during an execution of the program.*

We call the meaning of the program the *semantics* of the program. In this paper we are concerned with the following semantics, called the collecting semantics [2].

Definition 2. *Define the collecting semantics as a set of all possible program states that may occur for some execution and some input.*

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

¹ See Cousot lecture notes: <http://www.cs.wisc.edu/~cs704-1/LectureNotes/9.AbstractInterpretation.txt>

A key idea in program verification is that the collecting semantics can be approximated using a terminating program that takes as input the program and some approximation parameters and gives either an under-approximation or an over-approximation, which we define as follows.

Definition 3 (Over-Approximation). Let S be the semantics of a program. We say that any P^l where $S \subseteq P^l$ is an over-approximation.

Definition 4 (Under-Approximation). Let S be the semantics of a program. We say that any P_l where $P_l \subseteq S$ is an under-approximation.

The approximation is often useful to check certain concerns about the program. These concerns are expressed as some conditions called *error states* that need to be avoided by the program to be considered correct.

If the over-approximation does not contain the error states, then the program is considered correct. However, error states contained in an over-approximation may be spurious. Hence they do not prove the program incorrect.

The spurious error states may be avoided by tightening the over-approximation. If repeated tightening fails to eliminate the error states, then we may suspect that the program is incorrect. By using an under-approximation, we can often prove that the program is incorrect, i.e., falsify it. If an *under-approximation* of the semantics contains some error state, then the program is incorrect. Falsification identifies some of the errors in the program, hence it is a useful aid in debugging the program.

Our program verification approach falls into the category of *Abstract Interpretation*. The *abstract interpretation* technique provides the framework for extracting abstract collecting-semantics of a program [2,3,4,5,6,7,8]. The abstraction usually over-approximates the values of the program variables in a convex state space that models all possible program states and makes verification of correctness possible. Many different abstractions and combinations of abstractions evolved over the years and hence no succinct definition of abstract interpretation exists [2]. Rather, abstract interpretation gathers information about programs in order to provide sound answers to questions about their run-time behaviors. These semantics can then be used to design automatic program analyzers [5]. Abstract interpretation is often understood in terms of *abstract-evaluation* using an *abstract interpreter* on an *abstraction* of the program.

Definition 5 (Semantic operator). Let P be a program written in a language L . Define the semantic operator S as a mapping from L to the semantic domain of L denoted D .

$$S : L \rightarrow D \quad (1)$$

We usually write the semantics of P as $S[P] \in D$.

Definition [2] gives an example representation for the semantic domain.

Definition 6 (Abstraction operator). The abstraction of a program P written in a language L maps the semantics of P in D to an abstract domain $D^\#$

$$\alpha : D \rightarrow D^\# \quad (2)$$

² See www.di.ens.fr/~cousot/AI/ for Cousot's overview of Abstract Interpretation.

The purpose of the abstract domain is to provide a decidable domain of that can be used to evaluate the program.

Definition 7 (Abstract Evaluation). *Given a program P and an abstraction operator α , $\alpha[S[P]]$ is an abstract evaluation if it halts with an over-approximation of P .*

In the constraint database approach we perform abstract evaluation by creating a Datalog query for a constraint database and execute the query using constraint database approximation techniques.

The constraint database approach to program verification is a widely applicable method similar to abstract interpretation. However, while abstract interpretation methods usually rely on widening operators that yield convex approximations, the constraint database approach can yield non-convex approximations. This extra precision still allows an efficient calculation of approximate program semantics, which are crucial to the problem of program verification.

The remainder of the paper is organized as follows: Section 2 gives a review of constraint database approximation techniques. Section 3 describes the constraint database approach to verifying programs. Section 4 gives experimental results for three sample programs and uses them as a comparison to other techniques. Section 5 discusses the running time of the method and gives the running time for verifying each of the sample programs. Finally, Section 6 gives conclusions and future work.

2 Review of Constraint Database Approximation

The *constraint logic programming* languages proposed by Jaffar and Lassez [9], whose work led to CLP(R) [10], by Colmerauer [11] within Prolog III, and by Dincbas et al. [12] within CHIP, were Turing-complete. Kanellakis, Kuper, and Revesz [13] considered those to be impractical for use in database systems and proposed less expressive *constraint query languages* that have nice properties in terms of guaranteed and efficient evaluations. Many researchers advocated extensions of those languages while trying to keep termination guaranteed. For example, the least fixed point semantics of Datalog (Prolog without function symbols and negation) with integer gap-order constraint programs can always be evaluated in a finite constraint database representation [14].³

Definition 8 (Addition Constraints). *Addition constraints [15] have the form:*

$$\pm x \pm y \theta b \text{ or } \pm x \theta b \tag{3}$$

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

By allowing addition constraints, it is easy to express a Datalog program that will not terminate using a standard bottom-up evaluation [15]. Consider the following Datalog program:

³ A gap-order is a constraint of the form $x - y \geq c$ or $\pm x \geq c$ where x and y are variables and c is a non-negative integer constant.

$$\begin{aligned}
D(x, y, z) & :- x - y \leq 0, \quad -x + y \leq 0, \quad z \leq 0, \quad -z \leq 0. \\
D(x, y, z) & :- D(x', y, z'), \quad x - x' \leq 1, \quad -x + x' \leq -1, \\
& \quad z - z' \leq 1, \quad -z + z' \leq -1.
\end{aligned} \tag{4}$$

This expresses that the *Difference* of x and y is z . Further, based on Equation (4) we can also express a *Multiplication* relation as follows:

$$\begin{aligned}
M(x, y, z) & :- x \leq 0, \quad -x \leq 0, \quad y \leq 0, \quad -y \leq 0, \quad z \leq 0. \\
M(x, y, z) & :- M(x', y, z'), \quad D(z, z', y), \quad x - x' \leq 1, \quad -x + x' \leq -1 \\
M(x, y, z) & :- M(x, y', z'), \quad D(z, z', x), \quad y - y' \leq 1, \quad -y + y' \leq -1
\end{aligned} \tag{5}$$

Using Equations (4) and (5) we can express Diophantine equations which by [16] is Turing complete. Hence, in the limit as $l \rightarrow -\infty$, this method is Turing complete and can express any program. In this paper we limit ourselves to verifying programs with integer variables.

The D and M recursive programs must be evaluated until no new facts are discovered. This leads to the well known least fixed point definition.

Theorem 1 (Tarski's fixed point Theorem [17]). *Let (L, \subseteq) be any complete lattice. Suppose $F : L \rightarrow L$ is monotone increasing. Then the set of all fixed points of f is a complete lattice with respect to \subseteq .*

Let F be the set of facts discovered after evaluation of the Datalog rule given in Equation (4). Repeated application of $D^n(F)$ will not reach a point where it has discovered all the facts. If $D^n(F) = D^{n+1}(F)$, D will have reached a least fixed point.

Definition 9 (Least Fixed Point). *The least fixed point of a function f is a fixpoint v such that v is smaller than or equal to every other fixpoint of f .*

However the programs from (4) and (5) will never reach a fixed point. For example applying the recursive rule D gives the following facts where the bound on the right continues to increase to infinity.

$$\begin{aligned}
D(x, y, z) & :- x - y = 0, \quad z = 0. \\
D(x, y, z) & :- x - y = 1, \quad z = 1. \\
D(x, y, z) & :- x - y = 2, \quad z = 2. \\
& \dots
\end{aligned}$$

For Datalog, we always have a least fixed point [18], but when we add addition constraints there must be an approximation method that terminates the evaluation to find an approximation.

For constraint databases, Revesz [19,15] introduced two methods for approximating the least fixpoint evaluation of addition constraints by modifying the standard bottom-up evaluation.

Definition 10 (Lower-Bound Modification). *Let $l < 0$ be any fixed integer constant. We change in the constraint tuples the value of any bound b to be $\max(b, l)$. Given a Datalog program P the result of a bottom-up evaluation of P using this modification is denoted P_l .*

Definition 11 (Upper-Bound Modification). Let $l < 0$ be any fixed integer constant. We delete from each constraint tuple any constraint with a bound that is less than l . Given a Datalog program P the result of a bottom-up evaluation of P using this modification is denoted P^l .

Example 1 (Lower/Upper-Bound Modification). Consider the difference relation D and suppose that we set an approximation bound at $b = -2$. The lower bound approximation changes in the constraint tuple the value of any bound b to be $\max(b, u)$. This value will not cause the evaluation given in Equation (6) to change until the bound a bound is equal to 3. Perform the evaluation as follows:

$$D(x, y, z) \text{ :- } x' - y \leq 2, \quad -x' + y \leq -2, \quad -z' \leq 2, \quad -z' \leq -2 \quad (6)$$

$$x - x' \leq 1, \quad -x + x' \leq -1, \quad z - z' \leq 1, \quad -z + z' \leq -1.$$

Simplifying results in bounds that cause a problem.

$$D(x, y, z) \text{ :- } x - y \leq 3, \quad -x + y \leq -3, \quad z \leq 3, \quad -z \leq -3 \quad (7)$$

Applying the lower bound rule, change the any bound greater than 3 to be 2. This results in the following:

$$D(x, y, z) \text{ :- } -x - y \leq 2, \quad -x + y \leq -3, \quad z \leq 2, \quad -z \leq -3 \quad (8)$$

The last step in the requires checking the satisfiability of the modified clause. Combining the first two and last two constraints results in:

$$D(x, y, z) \text{ :- } 0 \leq -1 \quad (9)$$

Since Equation (9) is not satisfiable, the evaluation does not add a clause to the relation, and the evaluation of the recursive clause halts.

Applying the upper bound rule, to Equation (7), requires that we delete the second and fourth constraints which gives:

$$D(x, y, z) \text{ :- } x - y \leq 3, \quad z \leq 3 \quad (10)$$

This fact is added to the relation and we use it in the recursive call:

$$D(x, y, z) \text{ :- } x' - y \leq 3, \quad z' \leq 3$$

$$x - x' \leq 1, \quad -x + x' \leq -1, \quad z - z' \leq 1, \quad -z + z' \leq -1.$$

The result of this is

$$D(x, y, z) \text{ :- } x - y \leq 4, \quad z \leq 4 \quad (11)$$

This constraint is not added to the relation because it is subsumed by (10). Trying all the other facts in the recursive clause also results in a subsumed fact and hence the evaluation halts.

These modifications lead to the following approximation theorem.

Theorem 2 (Revesz [15]). *For any Datalog program P and constant $l < 0$ the following is true:*

$$P_l \subseteq \text{lfp}(P) \subseteq P^l \quad (12)$$

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

Hence l can be considered a parameter that controls how tight the over/under-approximation will be.

We implemented these two constraint modifications in CDB-PV to allow the over-approximation and under-approximation of the semantics of Datalog with addition constraints for program verification.

3 The Constraint Database Approach to Verification

CDB-PV is built on the MLPQ [20] constraint database system which provides a high degree of precision by allowing non-convex and disjoint regions to represent collecting semantics. We control the level of approximation by a single parameter l which corresponds to the parameters used in Definitions 10 and 11. Figure 1 provides an overview of the constraint database approach to the verification of programs [21].

The first two steps form the framework that translates a program into Datalog. The next step calculates an over-approximation given the bounding parameter l . The results from the over-approximation (or under-approximation) often contain a large set of data due to the disjoint representation of variable values. The constraint database approach simplifies interpretation of results by providing native facilities to query the results for error states using Datalog or SQL. Not finding the error state in the over-approximation verifies program correctness. If we suspect that the error state is present, we perform the under-approximation. Finding the error state in the under-approximation falsifies the program.

In theory the constraint database approach can reach arbitrary precision by calculating the under-approximation and over-approximation repeatedly as $l \rightarrow -\infty$. Hence this method approaches a precise evaluation. While this may not be possible for every constraint in an invariant, it may be reasonable to lower l to a point where the constraint in the over-approximation and under-approximation that identifies an error state converges. This method provides a way to find constraints in invariants that converge in parallel with constraints that do not even though a precise evaluation is not possible in general.

The framework for translating a program to Datalog adapts the standard pre-condition transition system used in static analysis and compiler optimization techniques.

Definition 12 (Transition System). *A transition system is a tuple $(S, \Lambda, \rightarrow)$ where S is a set of states, Λ is a set of labels and $\rightarrow \subseteq S \times \Lambda \times S$ is a ternary relation of labeled transitions. If $p, q \in S$ and $\beta \in \Lambda$, then $(p, \beta, q) \in \rightarrow$ is written as:*

$$p \xrightarrow{\beta} q$$

where β is a set of conditions and operations to the source state variables that must be made to enter the target state.

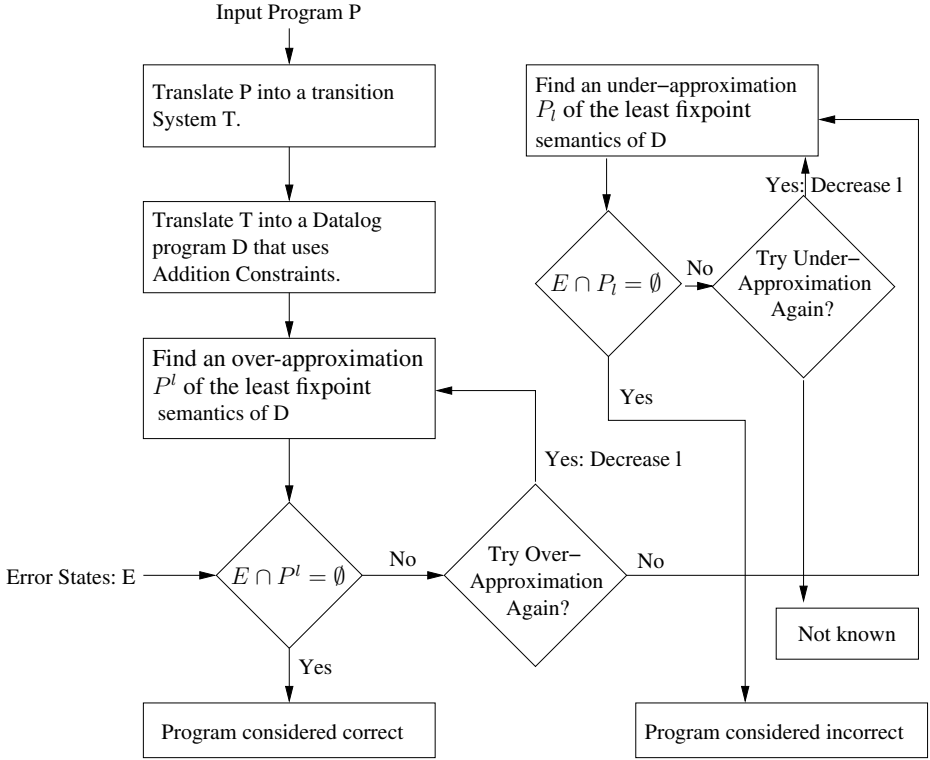


Fig. 1. Constraint Database Approach

The abstract domain we use consists of addition constraints. The framework translates a program into Datalog in the first two steps shown in Figure 1.

Given a program P with n lines of code and m variables, step (1) gives the transition system where a program statement (or state) $p_i \in S$ denotes the program statement on line i about to be executed. A transition from some state p_j to p_i denoted $p_j \xrightarrow{\beta} p_i$ represents the rule to enter state p_i where β contains the “execution” of the program statement on line j . The values changed by β affect the values available for execution in p_i . *How the new values affect the values of variables in state p_i will determine the type of approximation.* In Datalog these values will be added to the set of existing values. In abstract interpretation the new values cause widening of the existing invariants.

For example, Miné [22] defines an abstract interpretation widening technique as follows:

Definition 13 (Widening Operator of [22]). Let M and N be two ABMs. Then the widening of M by N , written as $M \nabla N$ is defined as:

$$[M \nabla N][i, j] = \begin{cases} M[i, j] & \text{if } M[i, j] \leq N[i, j] \\ -\infty & \text{if } N[i, j] \leq M[i, j] \end{cases}$$

Example 2. The simple program shown in Table 1 is translated into Datalog where transition system is given in Figure 2. The constraint relations L2 - L7 shown at the right represent the variable values prior to the execution of the corresponding lines at the left. L1 remains undefined because no variables have been assigned prior to the execution of Line 1.

Table 1. Simple Goto Program

		<code>begin%RECURSIVE%</code>
1. <code>a ← 0</code>		<code>L2(a) :- a=0.</code>
2. <code>a ← a + 1</code>		<code>L2(a) :- L5(a).</code>
3. <code>if a > 2 then goto 6</code>		<code>L3(a) :- L2(a1), a-a1=1.</code>
4. <code>if a = 2 then goto 7</code>	→	<code>L4(a) :- L3(a), a≤2.</code>
5. <code>goto 2</code>		<code>L5(a) :- L4(a), a<2.</code>
6. <code>...</code>		<code>L5(a) :- L4(a), a>2.</code>
7. <code>...</code>		<code>L6(a) :- L3(a), a>2.</code>
		<code>L7(a) :- L4(a), a=2.</code>
		<code>end%RECURSIVE%</code>

4 Experiments and Results

We tested the constraint database approach on three different examples. The first example compares our constraint database approach the widening technique of Miné using the simple code example from Example 2. The second experiment gives a tight bound for the automaton from [6]. The final example demonstrates the verification of a search algorithm involving Euclidean distance calculations. We give an examination of running time for the method and each individual sample program in Section 5.

The left side of Table 2 shows invariants found by two abstract evaluation passes using the Miné widening technique given in Example 2. In the second entry, an invariant of $a \geq 1$ entering line (3) indicates that line (6) is executed.

We recursively evaluate the Datalog program in MLPQ with $l = -2$ and either over-approximation or under-approximation to obtain the result on the right in Table 2. The resulting invariants for line 3 never indicate that $a > 2$ and hence we find that the $L6(a)$ relation is missing. Suppose that line (6) identifies an error, then our program technique identifies the unentered error state correctly where the abstract interpretation method of Miné does not.

Consider the subway train speed regulation system in Figure 3 described by [6]. Each train detects “beacons” that are marks placed along the track and receives a “second” signal from a central clock.

Let b and s be counter variables for the number of beacons and second signals received. Further, let d be a counter variable that describes how long the train is decelerating

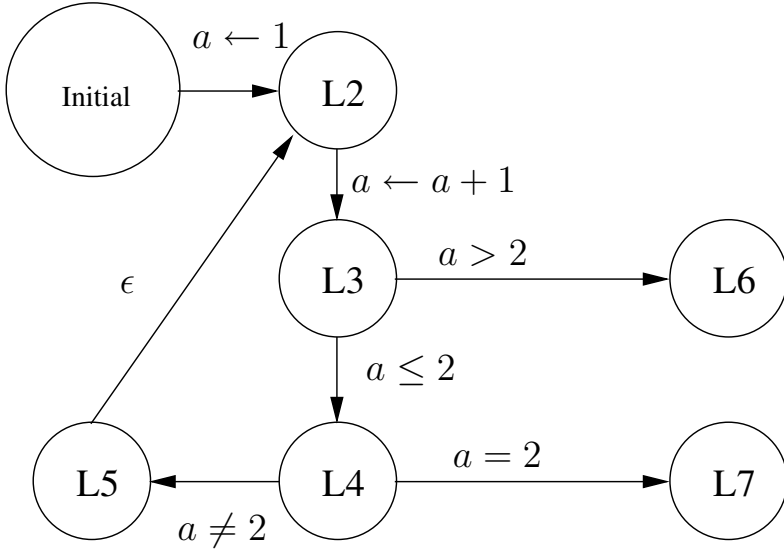


Fig. 2. Transition System for the Simple Program

Table 2. Invariants Obtained by Miné Widening

Miné Widening Results		
Line	1st Entry	2nd Entry
2	$0 \leq a \leq 0$	$0 \leq a$
3	$1 \leq a \leq 1$	$1 \leq a$ if condition $a > 2$ true goto 6
4	$1 \leq a \leq 1$	
5	$1 \leq a \leq 1$	

MLPQ output	
L2(a)	:- a=0.
L2(a)	:- a=1.
L3(a)	:- a=1.
L3(a)	:- a=2.
L4(a)	:- a=1.
L4(a)	:- a=2.
L5(a)	:- a=1.
L7(a)	:- a=2.

by applying its brake. The goal of the speed regulation system is to keep $|b - s| \leq 20$ while the train is running.

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

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

The counter automaton enforces the conditions described above using guard constraints followed by question marks, and $x++$ and $x--$ as abbreviations for the assignments $x := x + 1$ and $x := x - 1$, respectively.

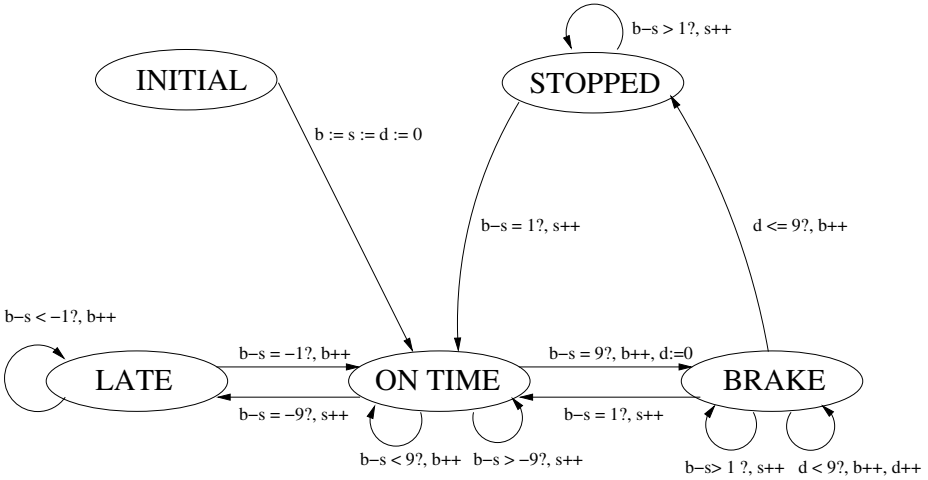


Fig. 3. Subway Automaton

The subway counter automaton from Figure 3 can be translated into the Datalog program shown in Table 3. It expresses the semantics (combinations of states and state variable values) of the automaton using difference constraints.

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

MLPQ Under Approximation

BRAKE	LATE	ONTIME	STOPPED
$1 \leq b - s \leq 19$	$-10 \leq b - s \leq -1$	$-9 \leq b - s \leq 9$	$1 \leq b - s \leq 20$
$10 \leq b \leq 19$	$10 \leq s \leq 19$	$0 \leq b \leq 9$	$11 \leq b \leq 20$
$0 \leq s \leq 18$	$0 \leq d \leq 9$	$0 \leq s \leq 18$	$0 \leq s \leq 9$
$0 \leq d \leq 9$		$0 \leq d \leq 9$	$0 \leq d \leq 9$

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

Further, since the above is a under approximation, any possible integer solution of the constraints below the state names *must occur* at some time. For example, the STOPPED relation must contain the case $b - s = 20$ at some time. Therefore, this automaton is incorrect by our earlier assumption.

The Verimag laboratory has software for testing program correctness using abstract interpretation. [6] gave the following over approximation derived using Verimag's software for the subway automaton.

Table 3. Subway Datalog Program

```

//Subway Automaton
begin%RECURSIVE%

ONTIME(b,s,d) :- b=0, s=0, d=0.
ONTIME(b,s,d) :- STOPPED(b,s1,d), b-s1=1, s-s1=1.
ONTIME(b,s,d) :- ONTIME(b1,s,d), b1-s<9, b-b1=1.
ONTIME(b,s,d) :- ONTIME(b,s1,d), b-s1>-9, s-s1=1.
ONTIME(b,s,d) :- ONBRAKE(b,s1,d), b-s1=1, s-s1=1.
ONTIME(b,s,d) :- LATE(b1,s,d), b1-s=-1, b-b1=1.
ONBRAKE(b,s,d) :- ONTIME(b1,s,d1), b1-s=9, b-b1=1, d=0.
ONBRAKE(b,s,d) :- ONBRAKE(b1,s,d1), d1<9, b-b1=1, d-d1=1.
ONBRAKE(b,s,d) :- ONBRAKE(b,s1,d), b-s1>1, s-s1=1.
STOPPED(b,s,d) :- ONBRAKE(b1,s,d), d<=9, b-b1=1.
STOPPED(b,s,d) :- STOPPED(b,s1,d), b-s1>1, s-s1=1.
LATE(b,s,d) :- ONTIME(b,s1,d), b-s1=-9, s-s1=1.
LATE(b,s,d) :- LATE(b1,s,d), b1-s<-1, b-b1=1.

end%RECURSIVE%

```

Verimag Over Approximation

BRAKE	LATE	ONTIME	STOPPED
$1 \leq b - s \leq d + 10$	$-10 \leq b - s \leq -1$	$-9 \leq b - s \leq 9$	$1 \leq b - s \leq 19$
$d + 10 \leq b$	$s \geq 10$	$b \geq 0$	$19 \leq 9s + b$
$0 \leq d \leq 9$		$s \geq 0$	$b \geq 10$

We showed in [23] that the Verimag system produced incorrect results. However a over-approximation still needs to be found to verify the automaton. We made several runs with different l values ranging from -10 to -30 for both over and under approximations. By increasing the l value to -20 alone and performing the evaluation with both approximations, we derive a tight bound $-10 \leq b - s \leq 20$ across the four constraint relations.

Suppose a yacht is traveling through the ocean between two ports. The yacht does not have enough supplies to make the trip, hence it must resupply at several possible locations. The program shown in Table 4 determines if a point $(22, 19)$ can be reached from a starting position of $(0, 0)$. It includes the `Depot` relation containing possible resupply locations. The `Leg` and `Reach` rules calculate Euclidian distance using the `D` (difference) and `M` (multiplication) relations. The approximation value l limits the evaluation of `D` and `M` which may be pre-computed to save time. The `reach` relation determines if the destination can be reached. This example can be extended by adding a relation for multiple destinations. In that case knowing error destinations would allow us to query the `reach` relation for incorrect values.

The results of running this program in MLPQ verify that the `reach` relation contains the values $x = 22$ and $y = 19$. This verifies the Resupply Depot program as correct.

Table 4. Yacht Resupply Example

```

begin%SupplyDepotOptimized%

Depot(id,x,y) :- id=1, x=0, y=19.
Depot(id,x,y) :- id=2, x=6, y=8.
Depot(id,x,y) :- id=3, x=15, y=12.
Depot(id,x,y) :- id=4, x=25, y=5.
D(x,y,z)      :- x-y=0, z=0.
D(x,y,z)      :- D(x1,y,z1), x-x1=1, z-z1=1.
D(x,y,z)      :- D(x1,y,z1), x-x1=-1, z0z1=-1
M(x,y,z)      :- x=0, y=0, z=0.
M(x,y,z)      :- M(x1,y,z1), D(z,z1,y), x-x1≥1, x1-x≥-1.
M(x,y,z)      :- M(x,y1,z1), D(z,z1,x), y-y1≥1, y1-y≥-1.
Leg(x,y)      :- x=0, y=0.
Leg(x,y)      :- Leg(x1,y1), Depot(id,x,y), AD(x,x1,dx),
                  AD(y,y1,dy), M(dx,dx,dx2), M(dy,dy,dy2),
                  dx2+dy2≤100, dx≤10, dy≤10.

Reach(x,y)    :- x=22, y=19, Leg(x1,y1), AD(x,x1,dx),
                  AD(y,y1,dy), M(dx,dx,dx2), M(dy,dy,dy2),
                  dx2+dy2≤100, dx≤10, dy≤10.

end%SupplyDepotOptimized%

```

5 Running Time of Methods and Sample Programs

Calculating the *lfp* of a Datalog program is exponential in the worst case. However, the running time depends on the amount of recursion in the Datalog program. We also note that a program need only be verified once and longer running times may be tolerated for program verification. The running times we report below indicate the time for the Datalog query to complete. The CDB-PV system uses 4,368kb of memory with no program loaded. The memory usage reported indicates the memory usage of the CDB-PV system when running the particular example and includes the 4,368kb. All runs were performed on an AMD Athlon 2000 with 1 GB or RAM except the Subway program which was run on an AMD X2 64bit computer with 1 GB of RAM.

The simple program from Table 2 can be evaluated precisely or with the under/over-approximation in the same running time. As this program has no recursion, the running time is less than our ability to measure (e.g. < 0.001 seconds) and the memory used is 5,556kb.

The yacht example from Table 4 depends on the D and M relations given in Equations (4) and (5). The under-approximation and over-approximation converge with

Table 5. Yacht Program Running Times (seconds) and Memory Usage (KB)

Bound	Under-Approximation		Over-Approximation	
	Time	Memory	Time	Memory
-10	1.969	5680	33.593	7020
-11	2.297	5716	37.886	7176
-12	2.922	5712	20.395	6716
-13	3.375	5724	25.014	7012
-14	3.966	5752	31.869	7336
-15	5.106	5764	35.538	7704

$l = -15$. We executed the query with $l = -10, \dots -15$. In this more complicated example we still have good running times and memory usage as shown in Table 6.

These results show a dip on the over-approximation at $l = -12$. We believe that runs with larger l values require more calculations to find the upper bound of the `Reach` relation. With $l < -12$, the calculation cost of D and M dominate the time and memory.

The subway Datalog program from Table 3 has more recursion and complicated calculations. We expect it to take the longest time. The running times and memory usage for various runs are given in Table 6.

Table 6. Subway Automaton Running Times (hh:mm:ss) and Memory Usage (KB)

Bound	Under-Approximation		Over-Approximation	
	Time	Memory	Time	Memory
-18	0:07:01	25,900	1:14:35	237,744
-19	0:09:32	31,028	1:15:50	231,596
-20	0:12:51	32,364	1:33:16	277,648
-21	0:16:35	43,308	1:43:28	286,264
-22	0:20:48	50,164	1:55:09	312,396

6 Conclusion and Future Work

We implemented an arbitrarily precise program verification and falsification method using constraint databases and approximation. The experiments showed that the constraint database method can more accurately approximate collecting semantics than other methods using widening techniques. Using over- and under-approximation we showed that our previous under-approximation constraint for the subway automaton is tight. In the ship resupply problem we showed our method is powerful enough to explore more complex mathematical expressions. While simple programs can be verified quickly, more complex programs may take longer as seen in the subway automaton verification. However, this method does provide precision beyond other techniques. Future work includes improving running time and memory efficiency while maintaining high precision and accuracy.

References

1. Kilroy, C.: Investigation: Air france 296 (1997), <http://www.airdisaster.com/investigations/af296/af296.shtml>
2. Cousot, P., Cousot, R.: Static determination of dynamic properties of programs. In: Proceedings of the Second International Symposium on Programming, 106–130 (1976)
3. Cousot, P., Cousot, R.: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In: Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of Programming Languages, pp. 238–252. ACM Press, New York (1977)
4. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, pp. 84–96. ACM Press, New York (1978)
5. Cousot, P., Cousot, R.: Abstract interpretation frameworks. *J. Log. Comput.* 2(4), 511–547 (1992)
6. Halbwachs, N.: Delay analysis in synchronous programs. In: Courcoubetis, C. (ed.) CAV 1993. LNCS, vol. 697, pp. 333–346. Springer, Heidelberg (1993)
7. Kerbrat, A.: Reachable state space analysis of lotos specifications. In: Proceedings of the 7th IFIP WG6.1 International Conference on Formal Description Techniques VII, London, UK, pp. 181–196. Chapman & Hall, Ltd., Sydney, Australia (1995)
8. Cousot, P.: Proving program invariance and termination by parametric abstraction, lagrangian relaxation and semidefinite programming. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 1–24. Springer, Heidelberg (2005)
9. Jaffar, J., Lassez, J.L.: Constraint logic programming. In: POPL '87: Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, New York, NY, USA, pp. 111–119. ACM Press, New York (1987)
10. Jaffar, J., Michaylov, S., Stuckey, P.J., Yap, R.H.C.: The CLP(R) language and system. *ACM Trans. Program. Lang. Syst.* 14(3), 339–395 (1992)
11. Colmerauer, A.: Note sur prolog iii. In: SPLT'86, Séminaire Programmation en Logique, 159–174 (1986)
12. Dincbas, M., Van Hentenryck, P., Simonis, H., Aggoun, A., Graf, T., Berthier, F.: The Constraint Logic Programming Language CHIP. In: Proceedings of the International Conference on Fifth Generation Computer Systems, vol. 2, pp. 693–702 (1988)
13. Kanellakis, P., Kuper, G., Revesz, P.: Constraint Query Languages. *Journal of Computer and System Science* 51(1), 26–52 (1995)
14. Revesz, P.: A Closed-Form Evaluation for Datalog Queries with Integer (Gap)-Order Constraints. *Theoretical Computer Science* 116(1-2), 117–149 (1993)
15. Revesz, P.: Introduction to Constraint Databases. Springer-Verlag, London (2002)
16. Matiyasevich, Y.V.: Hilbert's Tenth Problem. MIT Press, Cambridge (1993)
17. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math* 5(2), 285–309 (1955)
18. Ullman, J.: Principles of database and knowledge-base systems. Computer Science Press, Rockville, Md (1988)
19. Revesz, P.Z.: Reformulation and approximation in model checking. In: Koenig, S., Holte, R.C. (eds.) SARA 2002. LNCS (LNAI), vol. 2371, pp. 202–218. Springer, Heidelberg (2002)
20. Revesz, P., Chen, R., Kanjamala, P., Li, Y., Liu, Y., Wang, Y.: The MLPQ/GIS constraint database system. In: ACM SIGMOD International Conference on Management of Data (2000)

21. Revesz, P.: The constraint database approach to software verification. In: Cook, B., Podelski, A. (eds.) VMCAI 2007. LNCS, vol. 4349, pp. 329–345. Springer, Heidelberg (2007)
22. Miné, A.: The octagon abstract domain. In: Proceedings Analysis, Slicing and Transformation, pp. 310–319. IEEE Press, New York (2001)
23. Anderson, S., Revesz, P.: Verifying the incorrectness of programs and automata. In: Zucker, J.-D., Saitta, L. (eds.) SARA 2005. LNCS (LNAI), vol. 3607, pp. 1–13. Springer, Heidelberg (2005)

Generating Implied Boolean Constraints Via Singleton Consistency

Roman Barták

Charles University in Prague, Faculty of Mathematics and Physics
Malostranské nám. 2/25, 118 00 Praha 1, Czech Republic
roman.bartak@mff.cuni.cz

Abstract. Though there exist some rules of thumb for design of good models for solving constraint satisfaction problems, the modeling process still belongs more to art than to science. Moreover, as new global constraints and search techniques are being developed, the modeling process is becoming even more complicated and a lot of effort and experience is required from the user. Hence (semi-) automated tools for improving efficiency of constraint models are highly desirable. The paper presents a low-information technique for discovering implied Boolean constraints in the form of equivalences, exclusions, and dependencies for any constraint model with (some) Boolean variables. The technique is not only completely independent of the constraint model (therefore a low-information technique), but it is also easy to implement because it is based on ideas of singleton consistency. Despite its simplicity, the proposed technique proved itself to be surprisingly efficient in our experiments.

Keywords: implied constraints, reformulation, singleton consistency, SAT.

1 Introduction

Problem formulation is critical for efficient problem solving in formalisms like SAT (satisfiability testing), LP (linear programming), or CS (constraint satisfaction). LP and SAT formalisms are quite restricted, to linear inequalities in LP and logical formulas in SAT. Hence problem formulation is studied for a long time in LP and SAT because it is not always easy to express real-life constraints using linear inequalities or logical formulas. We can say that the problem formulation is the core of courses for normal users of LP and SAT, while the solving techniques are studied primarily by experts and researchers contributing to improving the solving techniques. Opposite to SAT and LP, the CS formalism is very rich concerning its expressivity (any constraint can be directly modeled there). Hence the users get a big freedom in expressing their problems as constraint satisfaction problems which has some negative consequences. First, because the solvers need to cover the generality of the problem formulation, it is hard to improve their efficiency, and, actually, we have not observed the dramatic increase of speed of constraint solvers similar to SAT and LP solvers. Second, the main burden on efficient problem solving is on the user who must understand the details of the solving process to formulate the problem in an

efficient way. Note that sometimes a small change in the model, such as adding a single constraint, may dramatically influence efficiency of problem solving which makes the modeling task even more complicated. There exist some rules of thumb how to design efficient constraint models [10,13], but constraint modeling is still assumed to be more art than science. There exist some automated techniques for on-fly problem re-formulation such as detecting and breaking symmetries during search (for a short survey see [13]) or no-good recording (introduced in [14] and formally described in [6]). Usually the problem (re-)formulation is up to the user by using techniques such as adding symmetry breaking or implied constraints, encoding parts of the problem using specialized global constraints, or adding dominance rules.

In this paper, we address the problem of fully automated generation of useful implied constraints in constraint satisfaction problems. Informally speaking, by a useful implied constraint we mean a constraint that is deduced from the existing model (hence implied) and that positively contributes to faster problem solving (hence useful). A fully automated technique means that the implied constraints are generated for any given constraint model without any user intervention. According to the principle that the best constraint model will be the one in which information is propagated first [10] we are trying to generate implied constraints that propagate more than the existing constraints (remove more inconsistencies from the model). Recall that more inconsistencies can be easily removed from any constraint model by applying a stronger consistency technique, for example by using path consistency instead of arc consistency. However, the main problem with stronger consistency techniques is their time and space complexity which disqualifies these techniques from being used repeatedly in the nodes of the search tree. Naturally, stronger consistency techniques can be applied once before the search starts but then their effect is limited to removing initially inconsistent values from variables' domains. We propose to exploit information from these stronger consistency techniques in the form of implied constraints that are deduced during the initial consistency process and added to the constraint model. In particular, we propose to use singleton arc consistency [5] to deduce new constraints between Boolean variables in the problem. The rationale for using singleton arc consistency (SAC) is that this meta-technique is easy to implement on top of any constraint model (singleton consistency is a meta-technique because it works on top of other "plain" consistency techniques such as arc consistency or path consistency). The reasons for restricting to Boolean variables are twofold. First, singleton consistency is an expensive technique especially when applied to variables with large domains so Boolean variables seem to be a good compromise. Second, we need to specify the particular form of constraints that we are learning, which is easier for Boolean variables. To be more specific, at this stage we are learning only the equivalence, implication, and exclusion constraints. In [1] we already showed that SAC over Boolean variables contributes a lot to removing initial inconsistencies so our hope is that the constraints derived from SAC can further help in problem solving.

The paper is organized as follows. After giving the initial motivation for our work, we will define more formally the used notions and techniques. Then we will present the core of the proposed technique and the paper will be concluded by an experimental section showing the benefits and detriments of the proposed method.

For now, we can reveal that despite the simplicity of the proposed method, the experiments showed surprising speed-ups for some problems.

2 Motivation

In [2] we proposed a novel constraint model for description of temporal networks with alternative routes similar to [4]. Briefly speaking, this model consists of a directed acyclic graph or in general a Simple Temporal Network [7], where the nodes are annotated by Boolean validity variables. There are special constraints between the validity variables describing logical relations between the nodes (we call them parallel and alternative branching). These constraints specify which nodes should be selected together to form one of the possible alternative routes through the network. Figure 1 shows an example of alternative branching together with a constraint model describing the relations between the validity variables.

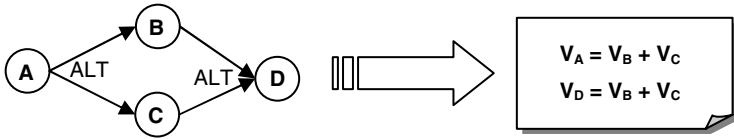


Fig. 1. A simple graph with alternative branching (left) and its formulation as a constraint satisfaction problem (left) over the validity variables

The above model is useful for description of manufacturing scheduling problems, but it suffers from several drawbacks. The main issue is that the problem of deciding which nodes are valid in the network is NP-complete in general [2]. Hence, opposite to Simple Temporal Networks [7] we cannot expect a complete polynomial constraint propagation technique that removes all inconsistencies from the constraint model. For example, the constraint model in Figure 1 cannot discover, using standard (generalized) arc consistency, that $V_A = 1$ if V_D is set to 1 (and vice versa). In [3] we proposed some pre-processing rules that can deduce implied constraints improving the filtering power of the constraint model. In particular, we focused on discovering (some) equivalent nodes, that is, the nodes whose validity status is identical in all feasible solutions (such as nodes A and D in Figure 1). Unfortunately, we also showed there that the problem whether two nodes are equivalent is also NP-hard. Our pre-processing rules from [3] are based on contracting the graph describing the problem and it is not easy to implement them and to extend them to other problems. Moreover, this method is looking only for equivalent nodes and ignores other useful relations such as dependencies and exclusions.

The above problem is not the only problem combining Boolean and temporal variables. Fages [8] studies a constraint model for describing and solving min-cutset problems and log-based reconciliation problems. Again, there are Boolean validity variables, which can be connected by dependency constraints in case of log-based reconciliation problems, and ordering variables describing the order of the nodes in a linear sequence of nodes (to model acyclicity of the selected sub-graph). We believe

that there are many other real-life problems where Boolean variables are combined with numerical variables. Our learning method might be useful for such problems to discover implied constraints between the Boolean variables that also take in account the other constraints. Naturally, we can learn implied constraints in problems with Boolean variables only, such as SAT problems.

3 Preliminaries

A *constraint satisfaction problem* (CSP) P is a triple (X, D, C) , where X is a finite set of decision variables, for each $x_i \in X$, $D_i \in D$ is a finite set of possible values for the variable x_i (the domain), and C is a finite set of constraints. A constraint is a relation over a subset of variables that restricts possible combinations of values to be assigned to the variables. Formally, a constraint is a subset of the Cartesian product of the domains of the constrained variables. We call the variable Boolean if its domain consists of two values $\{0, 1\}$ (or similarly $\{false, true\}$). A *solution to a CSP* is a complete assignment of values to the variables such that the values are taken from respective domains and all the constraints are satisfied. We say that a constraint C is (generalized) *arc consistent* if for any value in the domain of any constrained variable, there exist values in the domains of the remaining constrained variables in such a way that the value tuple satisfies the constraint. This value tuple is called a support for the value. Note that the notion arc consistency is usually used for binary constraints only, while generalized arc consistency is used for n-ary constraints. For simplicity reasons we will use the term arc consistency independently of constraint's arity. The CSP is *arc consistent* (AC) if all the constraints are arc consistent and no domain is empty. To make the problem arc consistent, it is enough to remove values that have no support (in some constraint) until only values with a support (in each constraint) remain in the domains. If any domain becomes empty then the problem has no solution. We say that a value a in the domain of some variable x_i is *singleton arc consistent* if the problem $P|x_i=a$ can be made arc consistent, where $P|x_i=a$ is a CSP derived from P by reducing the domain of variable x_i to $\{a\}$. The CSP is *singleton arc consistent* (SAC) if all values in variables' domains are singleton arc consistent. Again, the problem can be made SAC by removing all SAC inconsistent values from the domains. Figure 2 shows an example of a CSP and its AC and SAC forms.

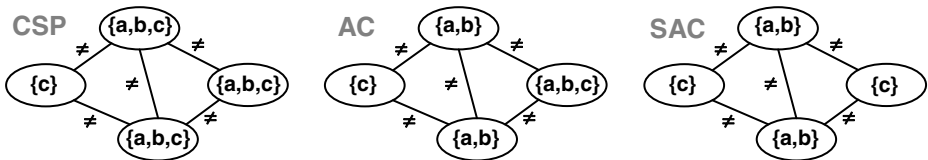


Fig. 2. A graph representation of a CSP, an arc consistent problem, and a singleton arc consistent problem (from left to right)

Assume now the constraint satisfaction problem with Boolean variables A, B, C, and D and with constraints $A = B + C$ and $D = B + C$ (like in Figure 1). This problem is both AC and SAC. Now assume that we assign value 1 to variable A. The problem

remains AC but it is not SAC because value 0 cannot be assigned to variable D. This is an example of weak domain pruning in our temporal networks with alternatives. If we now include constraint $A = D$ and make the extended problem AC then value 0 is removed from the domain of D by AC. Clearly, any assignment satisfying the original constraints also satisfies this added constraint. Hence we call this constraint *implied*, because the constraint is logically implied by the original constraints (sometimes, these constraints are also called *redundant*). Our goal is to find such implied constraints that contribute to stronger domain filtering.

4 Learning Via SAC

As we already mentioned in the introduction and motivation, our original research goal was to easily identify some equivalent nodes in the temporal networks with alternatives. Recall, that finding all equivalent nodes is an NP-hard problem [3] so we focused only on finding equivalences similar to those presented in Figure 1 (nodes A and D). An easy way, how to identify such equivalences, is a trial-and-error method similar to shallow backtracking or SAC. Basically, we will try to assign values to pairs of variables and if we find that only identical values can be assigned to the variables then we deduce that the variables are equivalent (they must be assigned to the same value in any solution). As a side effect, we can also discover some dependencies between the variables (if 1 is assigned to B then 1 must be assigned to A) and exclusions between the variables (either B or C must be assigned to 0 or in other words it is not possible to assign 1 to both variables B and C).

We will now present the learning method for an arbitrary constraint satisfaction problem P. Recall, that we will only learn specific logical relations between the Boolean variables of P. We will gradually try to assign values to variables and each time we try the assignment, this assignment is propagated to other variables (the problem is made AC). If the assignment leads to a failure then we know that the other value in the domain must be assigned to the variable (recall that we are working with Boolean variables). The whole learning process consists of two stages.

First, we collect information about which variables are instantiated after assigning value 1 to some variable A. We distinguish between *directly instantiated variables*, that is, those variables that are instantiated by making the problem $P|_{A=1}$ arc consistent (one value in the variable domain is refuted by AC so the other value is used), and *indirectly instantiated variables*, that is, those variables where we found their value by refuting the other value in a SAC-like style (AC did not prune the domain, but when we try to assign a particular value to the variable it leads to a failure so the other value is used). Informally speaking, if we assign value 1 to variable A and make the problem arc consistent then all variables that are newly instantiated are directly instantiated variables. Indirectly instantiated variables are those variables B that are not instantiated by AC in $P|_{A=1}$ but for which only one value is compatible with $A = 1$ because if the other value is assigned to B, it leads to a failure after making the problem AC (see procedure `LEARN` below). More formally, let B be a non-instantiated (free) Boolean variable in $AC(P|_{A=1})$, where $AC(P)$ is the arc consistent form of problem P (inconsistent values are removed from the domains of variables). If $P|_{A=1, B=0}$ is not arc consistent then value 0 cannot be assigned to B, hence value 1 must

be used for B. Symmetrically, we can deduce that value 0 must be assigned to B if $Pl_{A=1, B=1}$ is not arc consistent. Together, we can deduce which value must be used for B if value 1 is assigned to A. If both values for B are feasible then no information is deduced. If no value for B is feasible then value 1 cannot be used for A and hence A must be instantiated to 0. Note that information about indirectly instantiated variables is very important because it will help us to deduce implied constraints that improve propagation of the original constraint model. More formally, we are looking for implied constraints C such that $AC(Pl_C) \subset AC(P)$, where Pl_C is a problem P with added constraint C and the subset relation means that all domains in $AC(Pl_C)$ are subsets of relevant domains in $AC(P)$ and at least one domain in $AC(Pl_C)$ is a strict subset of the relevant domain in $AC(P)$. In other words, constraint C helps in removing more inconsistencies from problem P.

The learning stage deduces three types of implied constraints. If $B = 0$ is indirectly deduced from the assignment $A = 1$ and $A = 0$ is indirectly deduced from the assignment $B = 1$ then the pair $\{A, B\}$ forms an exclusion, which is an implied *exclusion constraint* ($A = 0 \vee B = 0$). Notice that this constraint really improves propagation because for example if 1 is assigned to A then the constraint immediately deduces $B = 0$, while the original set of constraints deduced no pruning for B. Similarly, if $B = 1$ is indirectly deduced from the assignment $A = 1$ then B depends on A, which is an implied *dependency constraint* ($A = 1 \Rightarrow B = 1$). Again, this constraint improves propagation. Note that we introduce this constraint only if variables A and B are not found to be equivalent. The equivalent variables are found using the following procedure. We construct a directed acyclic graph where the nodes correspond to the variables and the arcs correspond to the dependencies between the variables. These dependencies are found in the first stage, we assume both direct dependencies discovered by the AC propagation and indirect dependencies discovered by the SAC-like propagation. Strongly connected components of this graph form *equivalence classes of variables*. Note that if A and B are in a strongly connected component then $(A = 1 \Rightarrow^* B = 1)$ and $(B = 1 \Rightarrow^* A = 1)$, where \Rightarrow^* is a transitive closure of relation \Rightarrow . All equivalent variables must be assigned to the same value in any solution so we can put equality constraint between these variables.

The following code of procedure `Learn` shows both the data collecting stage and the learning stage of our method. `BoolVars(P)` is a set of not-yet instantiated Boolean variables in P, `doms(P)` are domains of P, $D_X = \{V\}$ means that the domain of variable X consists of one element V, and $AC(P)$ is the arc consistent form of problem P ($AC(P) = \text{fail}$ if problem P cannot be made arc consistent).

The main advantage of the proposed method is simplicity and generality. Thanks to meta-nature of singleton consistency it can be implemented easily in any constraint solver and it works with any constraint satisfaction problem (even if global constraints and non-Boolean variables are included). The time complexity of the data collection stage is $O(n^2 \cdot |AC|)$, where n is the number of Boolean variables and $|AC|$ is the complexity to make the problem arc consistent. Strongly connected components of the dependency graph can be found in time not greater than $O(n^2)$ and exclusions and dependencies are generated in time $O(n^2)$. Clearly, majority of time to learn implied constraints by the above method is spent by collection information using the SAC-like method.

```

procedure Learn (P: CSP)
  for each A in BoolVars(P) do // data collecting stage
    Q ← AC(P|A=1)
    Direct(A) ← { X/V | Dx = {V} in doms(Q)}
    for each B in BoolVars(Q) s.t. A ≠ B & Q ≠ fail do
      if AC(Q|B=0) = fail then
        Q ← AC(Q|B=1)
      else if AC(Q|B=1) = fail then
        Q ← AC(Q|B=0)
    end for
    Indirect(A) ← { X/V | Dx = {V} in doms(Q)} - Direct(A)
    if Q = fail then
      P ← AC(P|A=0)
      if P = fail then stop with failure
    end for
  // learning stage
  G ← (BoolVars(P), {(A,B) | B/1 ∈ Direct(A) ∪ Indirect(A)})
  Equiv ← StronglyConnectedComponents(G)
  Excl ← { {A,B} | B/0 ∈ Indirect(A) & A/0 ∈ Indirect(B)}
  Deps ← { (A,B) | B/1 ∈ Indirect(A) & ¬ {A,B} ⊆ X ∈ Equiv}
  return (Equiv, Excl, Deps)
end Learn

```

5 Implementation and Experiments

To evaluate whether our learning technique is useful for problem solving we implemented the learning technique in SICStus Prolog 3.12.3 and tested it on 1.8 GHz Pentium 4 machine running under Windows XP. Note that we used a naïve (non-optimal) implementation of the SAC algorithm that is called SAC-1 [5]. This algorithm simply assigns a value to the variable and propagates this assignment via standard arc-consistency algorithm. The algorithm does not pass any data structures between several runs which makes it non-optimal. Nevertheless, its greatest advantage is that the implementation is very easy and can be realized in virtually any constraint solver. For the experiments we used existing benchmarks for min-cutset problems [11] and a dozen of benchmarks for SAT problems [9].

5.1 Learning for CSP

In our first experiment, we compared efficiency of the original constraint model for min-cutset problems from [8] with the same constraint model enhanced by the learned implied constraints. Note that these constraint models contain both Boolean variables (validity) and integer variables (ordering of nodes). Recall that the min-cutset problem consists of finding the largest subset of nodes such that the sub-graph induced by these nodes does not contain a cycle. So it is an optimization problem. We used the data set from [11] with 50 activities and a variable number of precedence relations. Figure 3 shows the comparison of above models both in the runtime (milliseconds) and in the number of backtracks. It is important to say that the runtime

for the enhanced model consists of the time to learn the implied constraints and the time to solve the problem to optimality (using the branch-and-bound method). The time to learn the implied constraints is negligible there (from 80 to 841 milliseconds) and hence we do not show that time separately in the graphs. We used the well-known Brélaz variable ordering heuristic also known as dom+deg heuristic (the variables with the smallest domain are instantiated first, ties broken by preferring the most constrained variables).

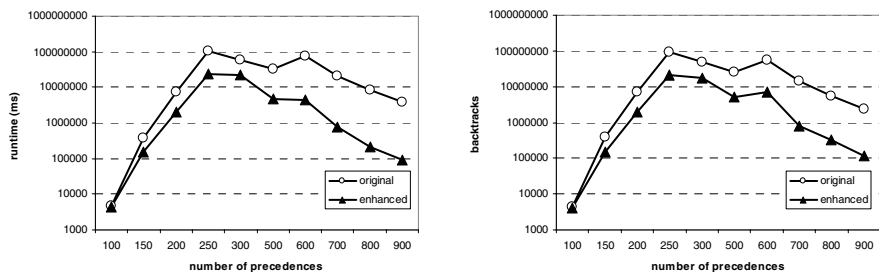


Fig. 3. Comparison of runtimes (milliseconds) and the number of backtracks for the original model of min-cutset problems and the model enhanced by the learned implied constraints with the Brélaz variable ordering heuristic

The graphs in Figure 3 show a significant decrease of the runtime and of the number of backtracks, which is a promising result especially taking in account that the time to learn is included in the overall time. This decrease is mainly due to the learned exclusion constraints which capture cycles in the graph (one node in the exclusion must be invalid to make the graph acyclic). Clearly, the Brélaz heuristic is also influenced by adding constraints to the model so the implied constraints may change the ordering of variables during search and hence influence efficiency. As we want to see also the effect of implied constraints on pruning the search space, we need to use exactly the same search procedure for both models. The straightforward approach is to use a static variable ordering. Figure 4 shows the comparison of both models using the static variable ordering heuristic. Again, we used the branch-and-bound method to solve the problem to optimality. Due to time reasons, we used a cut-off limit 300 000 000 milliseconds (>83 hours) for a single run so the most complex problems (200 - 300, and 600 precedences) were not solved to optimality for the original model and hence information about the number of backtracks is missing.

Again, the enhanced model beats the original model and shows a significant speedup. Moreover, by comparing both experiments, we can see that the learned constraints not only pruned more the search space by stronger domain filtering (which was our original goal) but in combination with the Brélaz heuristic they also make the search faster by focusing the search algorithm to critical (the most constrained) variables.

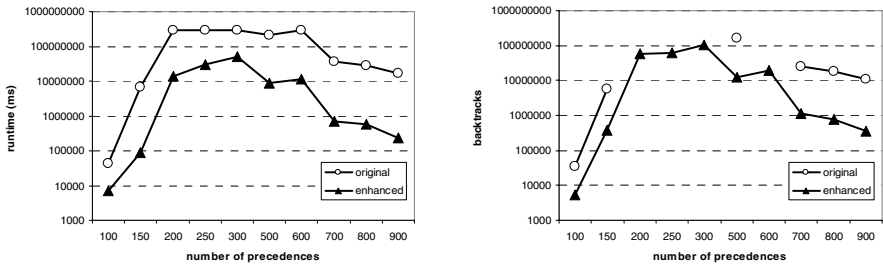


Fig. 4. Comparison of runtimes (milliseconds) and the number of backtracks for the original model of min-cutset problems and the model enhanced by the learned implied constraints with the static variable ordering heuristic (a logarithmic scale)

5.2 Learning for SAT Problems

Because our method works primarily with Boolean variables, the natural benchmark to test efficiency of the method was using SAT problems. We take several problem classes from [9], namely logistics problems from AI planning, all-interval problems, and quasigroup (Latin square) problems and encoded the problems in a straightforward way as CSPs. The choice of problem classes was driven by the idea that structured problems may lead to more and stronger implied constraints. It would be surely better to do more extensive tests with other problem classes, but a limited computation time forced us to select only few most promising classes. Again we used the Brélaz variable ordering heuristic in the search procedure which was backtracking search with maintaining arc consistency. Table 1 summarizes the results, it shows the problem size (the number of Boolean variables), the number of backtracks and the time to solve the problems (for the enhanced model the time includes both the time to learn as well as the time solve the problem), and the time for learning.

The experimental results show some interesting features of the method. First, the model enhanced by the learned implied constraints was frequently solved faster and using a smaller number of backtracks than the original model. The smaller number of backtracks is not that surprising, because the implied constraints contribute to pruning the search space. However, a shorter overall runtime for the enhanced model is a nice result, especially taking in account that the overall runtime includes the time to learn the implied constraints. The speed-up is especially interesting in the logistics problems, where the learning method deduced many exclusion constraints (probably thanks to the nature of the problem) which contributed a lot to decreasing the search space. The few examples when solving required more backtracks for the enhanced model (ais8, qg1-08, and qg7-13) can be explained by “confusing” the variable ordering heuristic by the implied constraints. Figure 3 and 4 showed that adding implied constraints influenced significantly the Brélaz variable ordering heuristic which is clear – the labeled variables have Boolean domains so the not-yet instantiated variables are ordered primarily by using the number of constraints in which they are involved. It may happen that in some problems this may lead to a wrong decision as no heuristic is perfect for all problems. It will be interesting to study further how the added implied constraints influence structure-guided variable ordering heuristics.

Table 1. Comparison of solving efficiency of the original and enhanced constraint models for selected SAT problems (the smallest #backtracks / runtime is in bold)

instance	size	original		enhanced		
		backtracks	runtime (ms)	backtracks	overall time (ms)	time to learn (ms)
logistics.a	828	>159827502	>60000000	4	53677	53657
logistics.b	843	>107546059	>60000000	38494	91622	65955
logistics.c	1141	>95990563	>60000000	26195	165537	150776
logistics.d	4713	>38809049	>60000000	5738102	28866167	16116604
ais6	61	16	10	3	400	390
ais8	113	178	120	523	3435	3155
ais10	181	3008	2914	118	14911	14811
ais12	265	66119	80386	140	49091	48921
qg1-07	343	26	811	0	146371	146291
qg1-08	512	331474	12445947	1791551	59608683	886605
qg2-07	343	34	1061	0	178987	178906
qg2-08	512	213992	8005862	213992	7980054	1053394
qg3-08	512	26	170	22	68018	67908
qg3-09	729	357521	2216758	25246	343845	233917
qg4-08	512	2956	12839	367	68088	66556
qg4-09	729	614	3925	86	225324	224934
qg5-09	729	1525	22573	0	1933	1933
qg5-10	1000	119894	2647697	0	61318	61318
qg5-11	1331	>1741008	>60000000	0	855000	854880
qg5-12	1728	>1195753	>60000000	0	6467810	6467810
qg5-13	2197	>802393	>60000000	41641	23622817	21695532
qg6-09	729	177	2304	0	51143	51113
qg6-10	1000	12234	238493	0	63732	63732
qg6-11	1331	1668478	34617658	4545	3233200	3153716
qg6-12	1728	>2512643	>60000000	586472	22669216	7159264
qg7-09	729	0	40	0	53337	53297
qg7-10	1000	348	6930	0	46557	46557
qg7-11	1331	27777	674701	0	429658	429658
qg7-12	1728	>2239230	>60000000	148648	10344354	6560683
qg7-13	2197	261	14101	525428	31893597	13893597

A second interesting feature is that for several quasigroup problems which have no feasible solution, the learning method proved infeasibility (in italics in Table 1) so no subsequent search was necessary to solve the problem. Again in most problems it was still faster than using the original constraint model. Finally, though we almost always improved the solving time, the overhead added by the learning method (the additional time to learn) was not negligible and the total time to solve the problem was sometimes worse than using the original model. This is especially visible in simple problems, where we spent a lot of time by learning, while in the meantime the backtracking search found easily the solution in the original model. This leads to a straightforward conclusion that if the original constraint model is easy to solve, it is useless to spent time by improving the model, for example by adding the implied constraints. Of course, the open question is how to find if the model is easy to solve.

5.3 Reformulation for SAT Solvers

In the previous section, we used SAT problems to demonstrate how the proposed learning method improves the solving time for these problems. However, we modeled the SAT problems using constraints and we used constraint satisfaction techniques to solve such models (combination of backtrack search and constraint propagation), which is surely not the best way to solve SAT problems. In the era of very fast SAT solvers, it might be interesting to find out if the implied constraints, that we learned using a constraint model, can also improve efficiency of the SAT solvers. We used one of the winning solvers in the SAT-RACE 2006 competition, RSat [12], to validate our hypothesis, that the learned constraints may also improve efficiency of SAT solvers. Table 2 shows the comparison of the number of backtracks, the number of decision (choice) points, and runtime for the original SAT problem and for the SAT problem with the added implied constraints. Again, we used the problem classes from [9].

There is clear evidence that the implied constraints decrease significantly the number of choice points of the RSat solver (and in most cases also the number of backtracks). This is an interesting result, because the RSat solver is using different solving techniques than the CSP solvers, to which our learning algorithm is targeted. Nevertheless, regarding the runtime the situation is different. Though the difference is not big, the model enhanced by the implied constraints is slower in most cases. This may be explained by the additional overhead for processing a larger number of

Table 2. Comparison of solving efficiency of the original model and the model with learned constraints solved by RSAT solver (the smallest #backtracks / #decisions / runtime is in bold)

instance	original			enhanced		
	backtracks	decisions	runtime (ms)	backtracks	decisions	runtime (ms)
logistics.a	137	1394	40	31	176	50
logistics.b	251	2019	60	119	558	90
logistics.c	238	2999	75	126	617	80
logistics.d	33	547	130	42	377	1022
ais6	14	46	5	0	11	0
ais8	20	74	10	0	22	10
ais10	1142	1877	90	0	37	20
ais12	19	152	25	0	56	30
qg1-07	105	134	140	44	72	130
qg1-08	4732	5608	1542	18288	20528	8142
qg2-07	35	54	130	37	53	130
qg2-08	14017	16270	6228	45678	52308	31320
qg3-08	122	175	40	122	153	50
qg3-09	57294	65736	26137	38434	44384	19027
qg4-08	638	737	100	586	667	110
qg4-09	8	30	60	6	23	60
qg5-11	44	78	230	0	4	370
qg5-13	38617	48396	36111	32971	38733	39046
qg6-09	0	15	70	0	3	130
qg6-12	12386	14426	7731	11171	13230	7761
qg7-09	1	7	70	0	3	130
qg7-12	4052	5042	1862	3360	4104	1912
qg7-13	2716	4139	1592	1375	1935	1131

clauses. Note that for some models, the percent of the implied constraints is 20-30% of the original number of constraints so if the solver is fast, this increase of the model size will surely influence the runtime. Still, it is interesting to see that the learned implied constraints are generally useful to prune the search space and perhaps, for more complicated problems, their detection may pay-off even if we assume time to learn these constraints (Table 1).

5.4 Learning Efficiency

The critical feature of the proposed method is efficiency of learning, that is, how much time we need to learn the implied constraints. In our current implementation, this time is given by the repeated calls to the SAC algorithm so the time depends a lot on the number of involved Boolean variables and also on the complexity of propagation (the number of constraints). The following figure shows the time for learning as a function of the number of involved Boolean variables for experiments from the previous sections (plus some additional SAT problems).

Clearly, due to the complexity of SAC, the proposed method is not appropriate for problems with a large number of Boolean variables. Based on our experiments, as a rough guideline, we can say that the method is reasonably applicable to problems with less than a thousand of Boolean variables. This seems small for SAT problems, but we believe it is a reasonable number of Boolean variables in CSP problems where non-Boolean variables are also included.

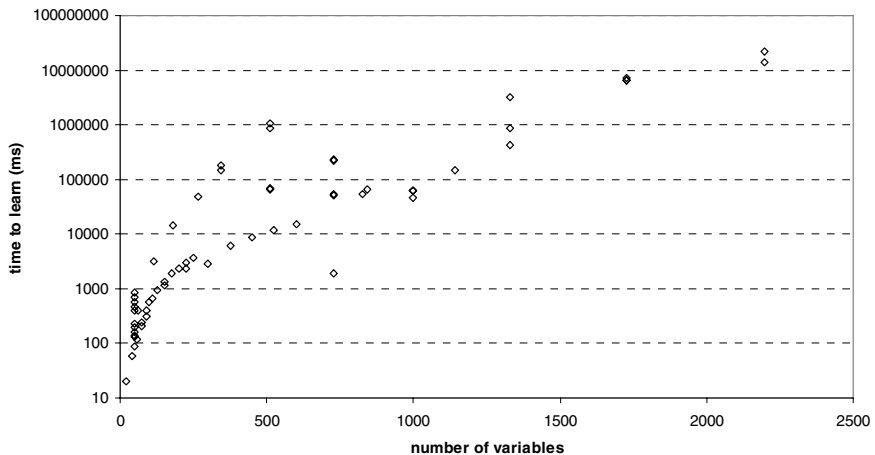


Fig. 5. Time to learn (in milliseconds) as a function of the number of involved Boolean variables (a logarithmic scale)

6 Conclusions

In the paper we proposed an easy to implement method for learning implied constraints over the Boolean variables in constraint satisfaction problems and we

presented some preliminary experiments showing a surprisingly good behavior of this method. In the experiments we used naïve hand-crafted constraint models, that is, the models that a “standard” user would use to describe the problem as a CSP, so the nice speed-up is probably partly thanks to weak propagation in these models. Nevertheless, recall the holly grail of constraint processing – the user states the constraints and the solver provides a solution. For most users, it is natural to use the simplest constraint model to describe their problem and we showed that for such models, we can improve the speed of problem of solving.

To summarize the main advantages of the proposed method: it is easy to implement, it is independent of the input constraint model, and it contributes to speed-up of problem solving. The experiments also showed the significant drawback of the method – a long time to learn (an expected feature due to using SAC techniques). Clearly, the method is not appropriate for easy-to-solve problems where the time to learn is much larger than the time to solve the original constraint model. On the other hand, we did the majority of experiments with the SAT problems where all variables are Boolean, while the method is targeted to problem where only a fraction of variables is Boolean, such as the min-cutset problem. We believe that the method is appropriate to learn implied constraints for the base constraint model which is then extended by additional constraints to define a particular problem instance. So learning is done just once while solving is repeated many times. Then the time to learn is amortized by the repeated attempts to solve the problem. The time to learn can also be decreased by identifying the pairs of variables that could be logically dependent. This may decrease the number of SAC checks. We did some preliminary experiments with the SAT problems, where we tried to check only the pairs of variables that are not “far each from other”, but the results were disappointing – the system learned fewer implied constraints. Still, restricting the number of checked pairs of variables may be useful for some particular problems.

Note finally that the ideas presented in this paper for learning Boolean constraints using SAC can be extended to learning other type of constraints using other consistency techniques. However, as our experiments showed, it is necessary to find a trade-off between the time complexity and the benefit of learning.

Acknowledgments. The research is supported by the Czech Science Foundation under the contract no. 201/07/0205.

References

1. Barták, R.: A Flexible Constraint Model for Validating Plans with Durative Actions. In: Planning, Scheduling and Constraint Satisfaction: From Theory to Practice. *Frontiers in Artificial Intelligence and Applications*, vol. 117, pp. 39–48. IOS Press, Amsterdam (2005)
2. Barták, R., Čepek, O.: Temporal Networks with Alternatives: Complexity and Model. In: *Proceedings of the Twentieth International Florida AI Research Society Conference (FLAIRS 2007)*, AAAI Press (2007)
3. Barták, R., Čepek, O., Surynek, P.: Modelling Alternatives in Temporal Networks. In: *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling (CI-Sched 2007)*, pp. 129–136. IEEE Press (2007)

4. Beck, J.Ch., Fox, M.S.: Scheduling Alternative Activities. In: Proceedings of the National Conference on Artificial Intelligence, pp. 680–687. AAAI Press (1999)
5. Debruyne, R., Bessière, C.: Some Practicable Filtering Techniques for the Constraint Satisfaction Problem. In: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI), pp. 412–417. Morgan Kaufmann, San Francisco (1997)
6. Dechter, R.: Learning while searching in constraint satisfaction problems. In: Proceedings of the Fifth National Conference on Artificial Intelligence, pp. 178–183. AAAI Press (1986)
7. Dechter, R., Meiri, I., Pearl, J.: Temporal Constraint Networks. *Artificial Intelligence* 49, 61–95 (1991)
8. Fages, F.: CLP versus LS on log-based reconciliation problems for nomadic applications. In: Proceedings of ERCIM/CompulogNet Workshop on Constraints, Praha (2001)
9. Hoos, H.H., Stützle, T.: SATLIB: An Online Resource for Research on SAT. In: SAT 2000, pp. 283–292. IOS Press, Amsterdam (2000), SATLIB is available online at www.satlib.org
10. Mariot, K., Stuckey, P.J.: Programming with Constraints: An Introduction. The MIT Press, Cambridge (1998)
11. Pardalos, P.M., Qian, T., Resende, M.G.: A greedy randomized adaptive search procedure for the feedback vertex set problem. *Journal of Combinatorial Optimization* 2, 399–412 (1999)
12. Pipatsrisawat, T., Darwiche, A.: RSat Solver, version 1.03. accessed in March (2007), <http://reasoning.cs.ucla.edu/rsat/>
13. Smith, B.: Modelling. A chapter in *Handbook of Constraint Programming*, pp. 377–406. Elsevier (2006)
14. Stallman, R.M., Sussman, G.J.: Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence* 9, 135–196 (1997)

Reformulating Constraint Satisfaction Problems to Improve Scalability

Kenneth M. Bayer¹, Martin Michalowski², Berthe Y. Choueiry^{1,2},
and Craig A. Knoblock²

¹ Constraint Systems Laboratory, University of Nebraska-Lincoln
{kbayer, choueiry}@cse.unl.edu

² University of Southern California, Information Sciences Institute
{martinm, knoblock}@isi.edu

Abstract. Constraint Programming is a powerful approach for modeling and solving many combinatorial problems, scalability, however, remains an issue in practice. Abstraction and reformulation techniques are often sought to overcome the complexity barrier. In this paper we introduce four reformulation techniques that operate on the various components of a Constraint Satisfaction Problem (CSP) in order to reduce the cost of problem solving and facilitate scalability. Our reformulations modify one or more component of the CSP (i.e., the query, variables domains, constraints) and detect symmetrical solutions to avoid generating them. We describe each of these reformulations in the context of CSPs, then evaluate their performance and effects in on the building identification problem introduced by Michalowski and Knoblock [1].

1 Introduction

Choueiry et al. [2] proposed to characterize a reformulation as a transformation of a problem \mathcal{P} from one encoding to another, where a problem is given by a *formulation* and a *query*, $\mathcal{P} = \langle \mathcal{F}, \mathcal{Q} \rangle$. The transformation may change the query and/or any of the components of the formulation. The goal of the transformation is to ‘simplify’ problem solving, where the benefit of the ‘simplification’ and other effects of the transformation are clearly articulated in the particular problem-solving context.

In this paper, we propose four reformulation techniques that operate on various aspects of a Constraint Satisfaction Problem (CSP) in order to improve the performance of problem solving. The problem formulation of a CSP is given by $\mathcal{F} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ where $\mathcal{V} = \{V_i\}$ is a set of variables, $\mathcal{D} = \{D_{V_i}\}$ the set of their respective domains, and \mathcal{C} a set of constraints. A constraint is a relation over a subset of the variables specifying the allowable combinations of values for the variables in its scope. A solution is an assignment to the variables such that all constraints are satisfied. The query is usually to find one satisfying solution, in which case the problem is in **NP**-complete in general. Alternatively, the query could be to find all possible solutions.

In this paper, we describe a reformulation of a CSP as a transformation of the original problem $\mathcal{P}_o = \langle \mathcal{F}_o, \mathcal{Q}_o \rangle$ into the reformulated problem $\mathcal{P}_r = \langle \mathcal{F}_r, \mathcal{Q}_r \rangle$, where \mathcal{F}_i indicates a formulation and \mathcal{Q}_i indicates a query, as illustrated in Figure 1. While our reformulations apply to a variety of resource allocation problems, in this paper we describe their application to the building identification problem (BID) proposed by Michalowski

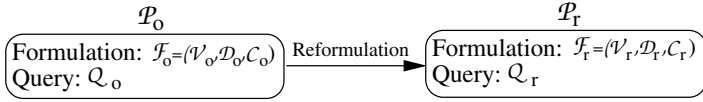


Fig. 1. The general pattern of a CSP transformation

and Knoblock [11]. The task is to assign a list of postal addresses to buildings appearing in a satellite image. A map provides the names of the streets and the positions of the buildings, but we do not know the addresses of the buildings or, for a building located on a street corner, on which street the building’s address lies. The original work established the feasibility of modeling and solving this problem as a CSP. However, their work did not scale well to larger problems [11]. We show that the reformulations we introduce allow us to solve larger problems. The largest problem solved by Michalowski and Knoblock included 34 buildings. In this paper, we scale up to problems involving 206 buildings.

This paper is structured as follows. In Section 2 we introduce a new type of global constraint and describe how to use symbolic values to reformulate the domains of variables in the scope of this constraint. In Section 3 we present a reformulated query that reduces runtime in practice. In Section 4 we describe how relaxing an assignment problem into a matching problem can be used to improve the performance of backtrack search used to solve the assignment problem. We also describe a symmetry detection process for generating all solutions to a maximum matching in a bipartite graph from a single solution to the matching. In Section 5 we apply these techniques to the building identification problem (BID). We present experimental results that demonstrate their effectiveness in Section 6. Section 7 discusses related reformulation work and Section 8 contains our conclusions and future research directions.

2 Domain Reformulation Using Symbolic Values

We propose ALLDIFF-ATMOST as a global constraint useful for resource allocation problems. In this section we first describe this constraint. We then discuss how to reformulate the domains of the variables in the scope of this constraint in order to reduce their size both for general and totally ordered domains. Section 5.2 illustrates the use of this constraint and its reformulation in a practical resource allocation problem.

2.1 ALLDIFF-ATMOST

Example 1. An emerging country received an aid to build 7 hospitals on its territory, but does not want to put more than 2 hospitals in areas with high volcanic activity.

We propose the constraint ALLDIFF-ATMOST to model this situation. Given a set of variables $\mathcal{A} = \{V_1, V_2, \dots, V_n\}$ with domains D_{V_i} , ALLDIFF-ATMOST(\mathcal{A}, k, d), where $d \subseteq D_{V_i}$, $k \in \mathbb{N}$, and $k \leq |d|$, requires that (1) all variables be different and (2) at most k variables in \mathcal{A} have values from d . Note that while the domains D_{V_i} may be different, d must be a subset of each one of them and D_{V_i} , and d and D_{V_i} may be finite or infinite.

Example 2. Consider with the variables $\mathcal{A}=\{V_1, V_2, V_3, V_4\}$ of a CSP, with $D_i=\{1, 2, \dots, 8\}$ and the constraint ALLDIFF-ATMOST($\mathcal{A}, 2, \{1, 3, 4, 5, 8\}$). The assignment $V_1 \leftarrow 5, V_2 \leftarrow 2, V_3 \leftarrow 7, V_4 \leftarrow 4$, and $V_5 \leftarrow 3$ satisfies the constraint.

2.2 ALLDIFF-ATMOST Reformulation

Our first reformulation technique applies to the ALLDIFF-ATMOST and the domains of the variables in its scope. The transformation, which we describe next, is theorem constant, in the sense that solutions to the reformulated problem map to solutions to the original problem [3]. The benefit of this reformulation is the reduction of the domain sizes. Because the complexity of many CP techniques depends on the sizes of the domains, the reformulation improves the solver performance.

We reformulate the domains of the variables in the scope of the constraint ALLDIFF-ATMOST(\mathcal{A}, k, d) by introducing k values s_l that we call *symbolic values* as follows:

$$\forall V_i \in \mathcal{A} \ D_{V_{i_r}} = \{s_1, s_2, \dots, s_k\} \cup (D_{V_i} \setminus d) \tag{1}$$

where the symbolic values s_j ($1 \leq j \leq k$) can take any distinct values in d . Applying this reformulation on Example 2 yields the following domains for all four variables: $D_{V_i}=\{s_1, s_2, 2, 6, 7\}$, where s_1, s_2 can take any different values in $\{1, 3, 4, 5, 8\}$. In Example 1, the domains become $\{s_1, s_2\} \cup \{\text{sites in non-volcanic areas}\}$ where s_1, s_2 are different and range over sites with volcanic activities.

This reformulation procedure operates on the problem formulation and affects both the ALLDIFF-ATMOST constraints and the domains of the variables in their scope, see Figure 2. However the most significant modification is the domain reformulation. We transform \mathcal{D}_o to \mathcal{D}_r , where in \mathcal{D}_r the domains of variables in \mathcal{A} are reformulated according to Equation (1). Replacing d in the original domains with k symbolic values reduces their sizes by $|d| - k$, which is useful when d is large or infinite.

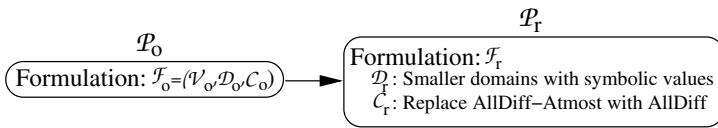


Fig. 2. The reformulation of ALLDIFF-ATMOST

This operation is particularly useful during backtrack search where the domain values are enumerated. If we want to assign ‘ground’ values to each symbolic value, we can do so as a post-processing step while ensuring that two symbolic values are always mapped back to distinct ground values. While a solution to the reformulated problem does not map to a unique solution to the original problem, we can generate any solution to the original problem from some solution to the reformulated problem.

Of particular concern is the interaction between this reformulation and the other constraints in the problem. When all the constraints in a problem can be checked on the symbolic values, as in the case of the BID, the reformulation is sound. When one or more constraints in a problem must be checked on the ‘ground’ values, then propagation

must run on the appropriate representation for each constraint and, as soon as domain filtering causes $|d| \leq k$, then reformulated domains should be dropped and ALLDIFF-ATMOST replaced with a ALLDIFF constraint.

2.3 Symbolic Intervals

When the values in the variables domains follow a total order, as in numeric domains, the domains are commonly represented as intervals and constraint propagation is typically restricted to the endpoints of these intervals, as in box-consistency algorithms. The reformulation of an ALLDIFF-ATMOST in the presence of totally ordered domains obviously remains valid. However, in order to *restrict propagation to the endpoints of the intervals* representing the domains, the following is needed:

1. We require the values in d to form a convex interval.
2. We must add ordering constraints between two consecutive s_i : $s_1 < s_2 < \dots < s_k$.
3. We must add total ordering constraints between the two extreme symbolic values, s_1 and s_k , and their closest neighbors in the reformulated domains, which is accomplished as follows. Let $D_{V_i}^l$ and $D_{V_i}^r$ be respectively the intervals of $D_{V_i} \setminus d$ to the left and right of, and adjacent to, d . The right endpoint of $D_{V_i}^l$ must be less than s_1 , and the left endpoint of $D_{V_i}^r$ must be greater than s_k . See Figure 3.

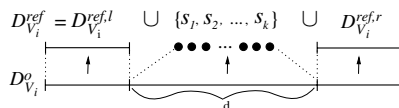


Fig. 3. ALLDIFF-ATMOST reformulation for totally ordered domains

4. When mapping the symbolic values back to ground values, the ground values must respect the total ordering imposed on the symbolic values.

Section 5.2 illustrates the use of ALLDIFF-ATMOST and its reformulation on the BID.

3 Query Reformulation

In a CSP, the query is usually to find a single solution (satisfiability problem) or all solutions (enumeration problem). However, in some applications, we may not be interested in entire solutions, but rather all the values that each variable takes in any solution. We call the problem corresponding to this query a *per-variable solutions*.¹ Thus, we reformulate the query from \mathcal{Q}_o , enumerating all solutions, to \mathcal{Q}_r , finding a per-variable solution. Formally, we define \mathcal{Q}_r as $\forall V_i, x \in D_{V_i}$, find if $\mathcal{P}_o \wedge (V_i \leftarrow x)$ is satisfiable. Figure 4 illustrates this reformulation. This reformulation changes the problem, because the solution returned will be different. However, in some cases, the per-variable solution is an acceptable alternative. This transformation changes the complexity class of the problem from a counting problem to a satisfiability one.

¹ Formally, this query corresponds to finding the minimal CSP.

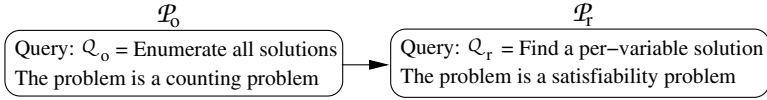


Fig. 4. Query reformulation

In this section we propose an algorithm to find per-variable solutions, and describe how this algorithm can be used to enforce high levels of relational consistency.

3.1 Per-variable Solutions

We can find the set of possible assignments to each variable in a CSP by solving the enumeration problem, that is finding all solutions and then iterating through the solution set to collect the values taken by each variable. However, the number of solutions to a CSP is $O(d^n)$, where n is the number of variables and d is the maximum domain size. Thus, finding all solutions may be prohibitively expensive.

We replace the query of finding all solutions (an enumeration problem) with the query of finding if a solution exists for every combination of variable-value pair (a polynomial number of satisfiability problems). Algorithm 1 tests for every variable-value pair (V_i, x) if the CSP with $V_i \leftarrow x$ is solvable. When a solution exist, x is added to the data structure returned by the algorithm. The algorithm returns the set of variables along with all their values that appear in a solution.

Input: $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$
Output: S , a per-variable solution

```

1 foreach  $V_i \in \mathcal{V}$  do
2    $S[V_i] \leftarrow \emptyset$ 
3 end
4 foreach  $V_i \in \mathcal{V}$  do
5   foreach  $x \in D_{V_i}$  do
6     if  $\mathcal{P}$  with  $V_i \leftarrow x$  has a solution then
7        $S[V_i] \leftarrow S[V_i] \cup \{x\}$ 
8     end
9   end
10  if  $|S[v]| = 0$  then
11    return  $\mathcal{P}$  has no solutions
12  end
13 end
14 return  $S$ 

```

Algorithm 1. Finding the per-variable solutions

The inner loop of the algorithm runs $O(nd)$ times. Each iteration requires determining the satisfiability of a CSP. This operation appears costly, but in cases where the original CSP has significantly more than nd solutions, Algorithm 1 can perform significantly better than enumerating all solutions to the CSP.

3.2 Relational Consistency

When solving a CSP, it is often beneficial to make the constraint network arc-consistent. Enforcing arc-consistency filters values from the variable domains that cannot exist in any solution to the problem. We can perform even more filtering by considering higher levels of consistency. Dechter and van Beek introduced *relational (i, m) -consistency* as the consistency of m non-binary constraints over every subset of i variables in the CSP [4]. Dechter [5] proposed the algorithm $RC_{(i,m)}$ for computing relational (i, m) -consistency. $RC_{(i,m)}$ works as follows. For every set C_m of m constraints in a constraint network, compute the join of the m constraints and project the result onto each subset of i variables. The algorithm is not practical for high values of m , because the memory requirements for computing and storing a join of m constraints rises exponentially with the number of variables.

Algorithm [1] computes a minimal network, which means that every value remaining in the network appears in at least one solution. Thus, the resulting network is the same as if we had executed $RC_{(1,m)}$. The difference between the two algorithms is that Algorithm [1] is polynomial space, whereas $RC_{(1,m)}$ is exponential space. We could generalize Algorithm [1] to consider sets of up to i variables rather than unique ones. This extension would allow the algorithm to produce the same results as $RC_{(i,m)}$, where the memory requirement rises with i , which quickly becomes impractical.

4 Constraint Relaxation for Problem Reformulation

At the core of many resource allocation problems lies the problem of matching between the elements of two sets: the tasks and the resources. In general, the problem may be complex (and likely intractable). However, in some cases, we may be able to identify constraints that can be removed to reduce the original problem into a matching problem in a bipartite graph.

Removing (or adding) a constraint in a problem formulation to yield a necessary (or sufficient) tractable approximation of the problem is a typical reformulation strategy. Examples abound and include: In AI, admissible heuristics generation for A* [6] and theory approximation [7]; in mathematical programming, linear relaxation of integer programs, Lagrangian relaxation [8], and the cutting-plane method.

In this section we first describe the relaxation of a resource allocation problem into a matching problem in a bipartite graph. Then, we describe techniques that take advantage of this reformulation when performing search for solving a CSP. Finally, we describe a symmetry detection technique that allows us to generate all the possible matchings in a bipartite graph from a single matching.

4.1 Matching as a Relaxation

Let $G = (X \cup Y, E)$ be a bipartite graph with edge set E , vertex set $V = X \cup Y$, and partitions X and Y , which are independent sets of vertices. We define a *match count* for each vertex in $v \in V$, which we denote $m(v)$, to be a positive (non-null) integer. A *matching* in G is a set of edges $M \subseteq E$ such that $\forall v \in V$ there exists at most one edge $e \in M$ incident to v . In this paper we consider a matching in G to be a set of edges

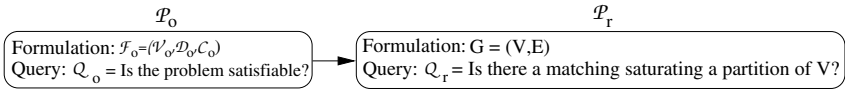


Fig. 5. Relaxation of a CSP as a matching problem

$M \subseteq E$ such that $\forall v \in V$ there exists at most $m(v)$ edges $e \in M$ incident to v . Further, we say that a matching M saturates vertex v iff M has exactly $m(v)$ edges incident to v ; and a matching M saturates a set S iff M saturates all vertices in S . Finding a matching that saturates S can be done in polynomial time (see Section 5.5).

We propose to reformulate a resource allocation problem by relaxing it into a matching problem in a bipartite graph that saturates one of the graph’s two partitions. Figure 5 illustrates this relaxation. While the original problem may be intractable, the reformulated one can be efficiently solved (i.e., in polynomial time). When the reformulated problem is not solvable, the more constrained original problem is not solvable. However, the solvability of the reformulated problem does not guarantee that of the original problem. Our reformulation is thus a necessary approximation [9].

4.2 Integrating the Matching Relaxation in Backtrack Search

When modeling a resource allocation problem as a CSP and solving it with backtrack search, we can take advantage of the relaxed problem in two ways:

1. As a *preprocessing step* prior to search, and
2. As a *lookahead mechanism* during search to filter out, from the domains of the future variables, those values that cannot yield a solution.

Prior to search, if we determine that the relaxed problem is not soluble, we can safely avoid using search. Further, during search, we can adapt the algorithm of Régis [10], which finds all edges of the bipartite graph that do not participate in *any* covering matching, to identify, in one step, all values in the domains of all future variables that do not participate in any saturating matching. This single operation allows us to filter the domains of all future variables in one step.

The reformulation into a matching problem is especially useful when finding per-variable solutions, because Algorithm 1 executes nd satisfiability tests. We propose to test, after line 5 in Algorithm 1, whether the relaxed problem is solvable, and proceed to line 6 only if this test succeeds. Otherwise, we return to line 5.

4.3 Generating Solutions by Symmetry

The set of maximum matchings in a bipartite graph can be obtained by enumerating all maximum matchings using an algorithm such as the one proposed by Uno [11]. In this section, we characterize all maximum matchings in a bipartite graph as symmetric to a single base matching, and proposed to use this symmetry to enumerate all solutions.

Our symmetry detection relies on two graph constructions described by Berge [12]: *alternating cycles* (AltCyc) and *even alternating paths starting at a free vertex* (EvAltP).

An AltCyc or EvAltP in a graph G relative to a matching M alternate between edges in M and edges not in M . If we take a maximum matching M and a AltCyc or EvAltP P , we can produce another maximum matching M' by computing the symmetric difference of M and P , denoted $M\Delta P$. We use that mechanism to identify all maximum matchings in a bipartite graph G as symmetric of a single maximum matching M . Let \mathcal{S} be the set of all AltCyc's and EvAltP's relative to M . We construct another maximum matching M_i by choosing a disjoint subset $\mathcal{S}_i \subseteq \mathcal{S}$ and computing $M\Delta\mathcal{S}_i$. M_i is symmetrical to M in that it is identical to M in all edges except those in \mathcal{S}_i . In fact, for any maximum matching M_j of G , we prove that there exists an \mathcal{S}_j such that $M_j = M\Delta\mathcal{S}_j$ using Lemma 3.1.9 of [13]. We generate \mathcal{S} by first orienting G using the construction described by Hopcroft and Karp [14]. From the oriented graph, we enumerate the alternating paths by finding all EvAltP's, as defined by Berge [12]. We enumerate the AltCyc's from the strongly connected components in the oriented graph as described by Régis [10]. Thus, to store the information necessary to enumerate all alternating paths and cycles, and therefore all maximum matchings, we only need to store a single base matching, the set of free vertices, and the set of strongly connected components [4].

Consider the bipartite graph $G = (X \cup Y, E)$, where $X = \{x_1, x_2, x_3, x_4\}$, $Y = \{y_1, y_2, y_3\}$, and $E = \{(x_1, y_1), (x_2, y_1), (x_2, y_2), (x_3, y_2), (x_3, y_3), (x_4, y_2), (x_4, y_3)\}$. Figure 6(a) shows a maximum matching M in G . $P = x_1y_1x_2$ is an alternating path and $C = x_3y_2x_4y_3x_3$ is an alternating cycle. We find other maximum matchings using the symmetric difference operator. Figure 6(b) show $M\Delta P$, Figure 6(c) shows $M\Delta C$, and Figure 6(d) shows $M\Delta(C \cup P)$.

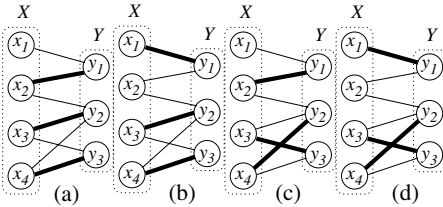


Fig. 6. Multiple matchings saturating Y

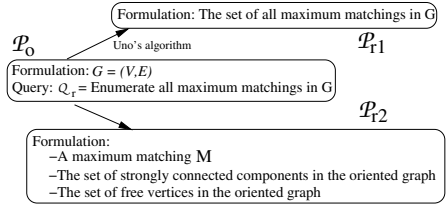


Fig. 7. Finding all maximum matchings

Figure 7 illustrates the two reformulations of \mathcal{P}_o , the problem of enumerating all maximum matchings. We can reformulate \mathcal{P}_o as \mathcal{P}_{r1} , the set of all maximum matchings, using Uno's algorithm. Alternatively, we can reformulate the problem as \mathcal{P}_{r2} , a base matching and its corresponding sets of strongly connected components and free vertices. All matchings can be enumerated from \mathcal{P}_{r2} as needed. Our construction has the same time complexity as Uno's, which is linear in the number of maximum matching. However, our characterization of the solutions as symmetries is a valuable one:

² An improvement suggested by a anonymous reviewer.

1. It provides a more compact representation of the set of solutions. Rather than storing all matchings, we store a single matching, a set of strongly connected components, and a set of free vertices.
2. In case one is indeed seeking *all*, or a given number of, the solutions to BID (similarly, to a resource allocation problem that has a maximum matching relaxation), we can generate every symmetric to that known single matching and test if it satisfies the additional constraints of the non-relaxed problem, when it does not, the matching is a solution to the non-relaxed problem found without search. Naturally, the number of maximum matchings can be large.

We do not currently exploit those features, but they deserve further investigations.

5 Application to the Building Identification Problem

We apply the four techniques presented above to the BID [11]. The task is to assign a list of addresses from a phone book to buildings appearing in a satellite image. Each address consists of the combination of a street name and a number. A map provides the names of the streets and the positions of the buildings. The map could come from an online source or a satellite image. We know the street names and the positions of the buildings, but we do not know the addresses of the buildings or, for a building located on a street corner, on which street the building’s address lies. A variety of data sources, such as a phone books, gazetteers, or property records, provide at least a partial list of addresses in a region. We generically refer to the addresses given as input as phone-book addresses regardless of their actual source. Figure 8 shows a BID instance.

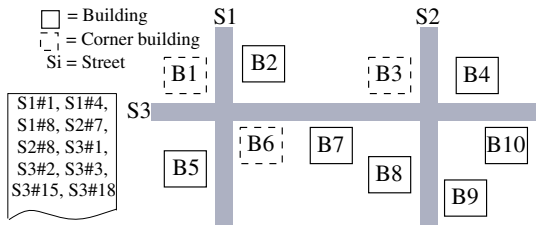


Fig. 8. An example of the BID problem

In general, the phone book may be incomplete, listing fewer addresses than there are buildings in the image, but the reverse does not hold. We must map every phone-book address to a building in the image. Michalowski and Knoblock established the feasibility of modeling and solving this problem as a CSP. However, their work did not scale well to larger problems. We show in Section 6 that the reformulations we propose allow us to solve larger problems.

5.1 CSP Model

Below we describe the variables and constraints in our CSP model of the BID, which improves on the one proposed in [11].

Our model uses three types of variables: *orientation* variables, *corner* variables, and *building* variables. In general, there are four Boolean *orientation variables*: *IncreasingNorth*, *IncreasingEast*, *OddOnEastSide*, and *OddOnNorthSide*. The first two are *ordering* variables and indicate whether or not addresses increase in value when moving toward the north and to the east. The remaining two are *parity* variables and indicate on which side of the street odd addresses occur. The *corner variables* represent the possible *streets* on which a corner building might be. We generate one corner variable for each corner building, whose domain is the list of streets on which the building could lie, and has size 2 in most cases. The corner buildings are natural ‘articulations’ in the constraint network: once the solver assigns values to all corner buildings, the constraint network degenerates into a set of chains (corresponding to buildings along street segments) that can be solved in a backtrack-free manner. Michalowski and Knoblock too noted this feature [11]. Thus, the solver instantiates corner buildings as soon as possible. The *building variables* represent the addresses (i.e., numbers) of the buildings. We generate a building variable for every building on the map. The domain of a variable is every possible address on the building’s streets.

Our model has five types of constraints: *parity*, *ordering*, *corner*, *phone book*, and *grid*. *Parity constraints* are binary constraints and ensure that the numbers assigned to buildings respect the values assigned to the parity (orientation) variables. *Ordering constraints* are ternary constraints, and link an ordering variable to two building variables along the same street. These constraints ensure that the addresses assigned to the building variables respect the ordering specified by the ordering variable. *Corner constraints* link the the corner and building variables of a corner building and ensure that the address assigned to the building is consistent with the street chosen for the building. *Phone-book constraints* exist for each street on the map. These constraints ensure that the solver assigns every address in the phone book to some building along that street. These constraints usually have a high arity, because their scope is the set of buildings along the street. *Grid constraints* exist between buildings across certain artificial grid-lines, depending on the region we are modeling. These constraints ensure that the addresses of adjacent buildings across the grid-lines are in separate numeric increments. For example, in many cities in the United States, addresses increase to the next increment of 100 across intersections.

5.2 Symbolic Values

If the phone book is incomplete, we must infer the missing numbers to add to the variables’ domains. Michalowski and Knoblock proposed to enumerate all numbers between 1 and the largest address that appears on the street [11]. Their approach has two problems. First, the choice of the upper limit is arbitrary. When the largest address is not in the phone book, this approach may yield incorrect solutions. The second problem with this approach is that the size of the domains becomes prohibitively large on real-world data. Using symbolic values in phone-book constraint solves both problems.

Let S be a street, P_S its set of phone-book addresses of a given parity, B_S the set of buildings on the side of S of that parity, and $[\min, \max]$ the range of address numbers on that side of S . The address numbers in P_S partition $[\min, \max]$ into consecutive convex intervals. In any such interval (i_1, i_2) , we cannot use more than $|B_S| - |P_S|$ addresses,

which we express as ALLDIFF-ATMOST($B_S, k_a, (i_1, i_2)$) with $k_a = \text{minimum}(|B_S| - |P_S|, \lfloor \frac{(i_2 - i_1) - 1}{2} \rfloor)$. We reduce the variables' domains using the reformulation of Section 2 on each of these intervals. For example, assume we have, on the even side of S , $B_S = \{B_1, B_2, \dots, B_5\}$, $P_S = \{S\#12, S\#18\}$, $\text{min}=2$, and $\text{max}=624$. An assignment cannot use more than 3 numbers in each of $[2, 12)$, $(12, 18)$, and $(18, 624]$, yielding 3 ALLDIFF-ATMOST constraints with the following arguments $(B_S, 3, [2, 12))$, $(B_S, 2, (12, 18))$, and $(B_S, 3, (18, 624])$. We replace the domain $[1, 624]$ of each variable with the significantly smaller set $\{s_1, s_2, s_3, 12, s_4, s_5, 18, s_6, s_7, s_8\}$ where $s_1, s_2, s_3 \in [2, 12)$, $s_4, s_5 \in (12, 18)$, and $s_6, s_7, s_8 \in (18, 624]$ and $s_i < s_j$ for $i < j$, see Figure 9. This process allows us to choose an arbitrarily large upper bound (max) on a given street.

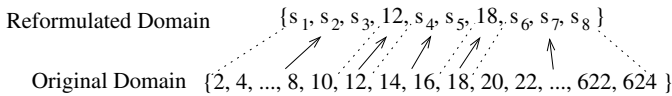


Fig. 9. Domain reformulation for the BID

5.3 Query Reformulation

Another challenge of real-world BID instances is the large number of solutions. If the phone book is incomplete, the problem is under-constrained, yielding a large number of solutions. One possible query would be to enumerate all the solutions to collect the acceptable list of addresses for each building [11]. By reformulating the query as proposed in Section 3, we can use Algorithm 1 to obtain the same result at a much cheaper cost. In summary, we replace the query: “Enumerate all solutions and collect the addresses taken by the buildings in these solutions” with the query “Find all the addresses that a given building can take.”

5.4 Constraint Relaxation for Problem Reformulation

We show below that, when no grid constraints exist, the BID problem can be modeled as a matching in a bipartite graph, and is thus tractable. The CSP approach of Michalowski and Knoblock remains pertinent in that it allows one to represent arbitrary street-addressing schemas used around the world, such as grid constraints. We propose the removal of grid constraints as a tractable relaxation of the BID.

Given an instance of this problem without grid constraints, we construct a bipartite graph $G = (B \cup S, E)$ as follows. First, assume an assignment to the orientation variables (there are 2^4 such assignments). For each building β in the problem, add a vertex b to B . For each street σ in the problem, add two vertices s_{odd} and s_{even} to S , one for each side of the street. For each building β , add an edge between vertex b and the street vertex corresponding to the street side on which β may be. (Note that corner buildings are on two streets.) Assuming odd numbers appear on the North and West sides of the street, Figure 10 shows the construction of G for the map in Figure 8. We can show that a matching in this graph that saturates S corresponds to a satisfactory assignment of streets to corner buildings.

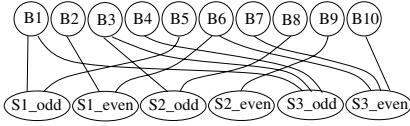
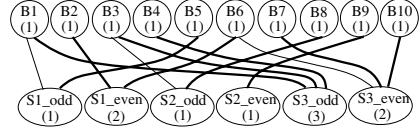

Fig. 10. Graph construction for Figure 8

Fig. 11. Satisfying matching for Figure 10

Figure 11 shows a satisfying matching for the graph from Figure 10, where the edges included in the matching are darkened. The numbers in parentheses indicate the match count of the vertex. This matching corresponds to an assignment of buildings to streets. Thus, we can now construct a solution to the problem as a post-processing step, where we simply assign numbers to each building along each street.

While the matching approach is powerful, it cannot model the grid constraint. Whether the problem with the grid constraints can be solved efficiently remains an open question. We propose to use the matching in 2 ways: (1) directly solve problems that have no grid constraints, and (2) use the relaxation to detect unsolvability (both as a preprocessing step and for lookahead as discussed in Section 4).

5.5 Solvers

It was simply impossible to solve any of our real-world data sets using the original query. We could solve them only with the reformulated query, using Algorithm 1.

We implemented two solvers: a matching-based solver and a search-based solver. The former finds a maximum matching using an $O(n^{5/2})$ algorithm by Hopcroft and Karp [4] after replacing each vertex in the bipartite graph by as many vertices as its match count. The latter uses backtrack search (BT) with nFC3, a look-ahead strategy for non-binary CSPs [5], and conflict-directed backjumping [6]. In BT, we implemented a hybrid representation of the domains: enumerated values and intervals. We use the interval representation to propagate ordering constraints (i.e., less-than constraints), and restrict this propagation to the bounds of the intervals without loss of pruning power. This representation improved our runtime by one order of magnitude.

When the problem given as input has no grid constraint, we use the matching solver in line 6 of Algorithm 1, which computes one matching in each of the nd loops. Thus, the solver runs in polynomial time, which is a significant improvement compared with the exponential-time backtrack search based solver. When the problem instance has grid constraints, we proceed as follows:

1. Preprocessing: We insert a call to the matching solver after line 5 in Algorithm 1 and proceed to line 6 if we find a matching, otherwise return to line 5.
2. Backtrack search (BT): We use the search-based solver in line 6.
3. Lookahead: In addition to nFC3, we filter in one step the domains of all future variables given the current path in the search (see Section 4).

Figure 12 illustrates the behavior of the solvers.

6 Experiments

Table 1 describes the properties of the regions of the city of El Segundo (CA), on which we ran our experiments. The number of calls refers to the total number of times to line 6 of Algorithm 1. Each call to line 6 was timed out after one hour. We report the number of timed out executions.

The completeness of the phone book indicates what percent of the buildings on the map have a corresponding address in the phone book. We created the complete phone books using property-tax data, and the incomplete phone books using the real-world phone-book.

Effect of domain reformulation. Table 2 shows the benefit of domain reformulation by comparing the performance when using the original domains or the reformulated domains. The experiment uses backtrack search (BT), but does not take into consideration the grid constraints. When the phone book is complete, no ALLDIFF-ATMOST constraints are present, and thus the reformulation does nothing. The advantage of the reformulation is clear when using the incomplete phone book.

Table 1. Case studies used in experiments

Case study	Phone book completeness	Number of			
		bldgs	cnrr	bldgs	blks
NSeg125-c	100.0%	125	17	4	4160
NSeg125-i	45.6%				1857
NSeg206-c	100.0%	206	28	7	4879
NSeg206-i	50.5%				10009
SSeg131-c	100.0%	131	36	8	3833
SSeg131-i	60.3%				2375
SSeg178-c	100.0%	178	46	12	4852
SSeg178-i	65.6%				2477

Table 2. Effect of domain reformulation

Case study	Avg. domain size		Runtime [sec]		Timeouts	
	Orig.	Ref.	Orig.	Ref.	Orig.	Ref.
NSeg125-i	1103.1	236.1	2943.7	744.7	0	0
NSeg206-i	1102.0	438.8	14818.9	5533.8	0	0
SSeg131-i	792.9	192.9	67910.1	66901.1	18	17
SSeg178-i	785.5	186.3	119002.4	117826.7	32	29

Table 3. Solvers' performance (no grid)

Case study	Runtime [sec]		
	BT	Matching	Matching + Symmetry
NSeg125-c	139.2	4.8	0.03
NSeg125-i	744.7	2.5	*
NSeg206-c	4971.2	16.3	0.06
NSeg206-i	5533.8	8.5	*
SSeg131-c	38618.3	7.3	0.26
SSeg131-i	66901.1	3.1	*
SSeg178-c	117279.1	22.5	0.41
SSeg178-i	117826.7	4.9	*

* Did not finish in 1 hour.

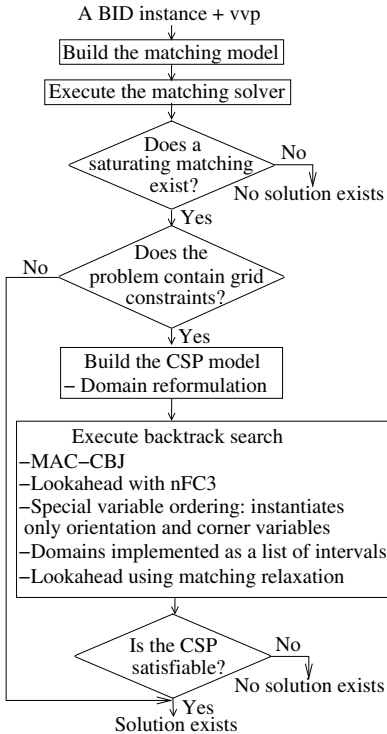


Fig. 12. Implementing Line 6 of Algorithm 1

Effect of query reformulation. As stated in Section 5.5, the sheer number of solutions made it impossible to solve problem instances with incomplete phone-books using the query of enumerating all solutions. Thus, without the query reformulation, we would not have been able to solve the incomplete phone-book instances.

Effect of finding symmetrical maximum matchings. In the absence of grid constraints, the BID can be solved in polynomial time by the matching solver. Here we compare backtrack search, a solver that uses Algorithm 1 with a matching solver, and a solver that uses the reformulation of symmetric matchings from section 4.3. Finding all symmetric matchings requires enumerating all matchings, which isn't feasible for the under-constrained incomplete phone-book problems. Thus, those problem instances timed out and are indicated by asterisks. However, when the number of solutions was small, such as when the phone-book is complete, the symmetry solver had significantly better performance than the per-variable matching solver. The benefit in terms of runtime reduction is shown in Table 3.

Effect of relaxing a CSP into a matching problem. To test the use of the matching relaxation as a preprocessing step and lookahead mechanism, we added grid constraints to each region. Table 4 shows the results of these experiments, comparing the performance of: (1) the backtrack search (BT), (2) BT with matching for preprocessing (Preproc+BT), (3) BT with matching for lookahead (Lkhd+BT), and (4) BT with matching for both purposes (Preproc+BT+Lkhd). We report runtime, number of timeouts, and number of calls to the CSP solver saved by the preprocessing. In all cases, the same solutions were found. Our results indicate that, in general, the integration of the matching and BT improves performance. There are exceptions, when the cost of the additional processing exceeds the gains in terms of reduced search space. However, even when we saw performance degradation, the degradation was minimal.

Table 4. Improvements due to preprocessing and lookahead

NSeg125-c + grid				SSeg131-c + grid			
BT	CPU [sec]	#Timeouts	Calls saved	BT	CPU [sec]	#Timeouts	Calls saved
Preprocessing+BT	100.8	0	-	Preprocessing+BT	17063.3	0	-
BT+Lkhd	33.2	0	97.0%	BT+Lkhd	5997.9	0	92.5%
Preproc+BT+Lkhd	140.2	0	-	Preproc+BT+Lkhd	9745.8	0	-
	39.6	0	97.0%		4256.0	0	92.5%
NSeg125-i + grid				SSeg131-i + grid			
BT	CPU [sec]	#Timeouts	Calls saved	BT	CPU [sec]	#Timeouts	Calls saved
Preprocessing+BT	1232.5	0	-	Preprocessing+BT	114405.9	30	-
BT+Lkhd	1159.1	0	62.6%	BT+Lkhd	114141.3	29	74.2%
Preproc+BT+Lkhd	726.6	0	-	Preproc+BT+Lkhd	107896.3	30	-
	701.1	0	62.6%		108646.5	30	74.2%
NSeg206-c + grid				SSeg178-c + grid			
BT	CPU [sec]	#Timeouts	Calls saved	BT	CPU [sec]	#Timeouts	Calls saved
Preprocessing+BT	2277.5	0	-	Preprocessing+BT	78528.6	14	-
BT+Lkhd	614.2	0	98.9%	BT+Lkhd	15717.9	1	91.9%
Preproc+BT+Lkhd	1559.2	0	-	Preproc+BT+Lkhd	74172.0	14	-
	443.8	0	98.9%		13961.1	1	91.9%
NSeg206-i + grid				SSeg178-i + grid			
BT	CPU [sec]	#Timeouts	Calls saved	BT	CPU [sec]	#Timeouts	Calls saved
Preprocessing+BT	4052.8	0	-	Preprocessing+BT	138404.2	35	-
BT+Lkhd	3806.7	0	87.8%	BT+Lkhd	103244.7	25	72.7%
Preproc+BT+Lkhd	3499.5	0	-	Preproc+BT+Lkhd	121492.4	32	-
	3510.0	0	87.8%		85185.9	22	72.7%

7 Related Work

Reformulation has been applied to a wide range of CSP problems with much success. The literature encompasses also approaches to modeling, abstraction, approximation, and symmetry detection. Nadel studied 8 different models of the n -Queens problem, some of which much easier to solve than others [17]. Glaisher proposed avoiding symmetry in the Eight Queens as far back as 1874 [18]. This topic has recently received increased attention, for example in the work of Puget [19] and Ellman [9]. Holte and Choueiry provide a general discussion on abstraction and reformulation in AI including CSPs [20]. Razgon et al. [21] introduced a class of problems they called Two Families of Sets constraints (TFOS), and a technique for reformulating TFOS problems into network flow problems. Conceptually, the relaxed problem we study in Section 4 constitutes a special case of the TFOS problem.

8 Conclusions and Future Work

We introduced four general reformulation techniques for CSP, and integrated them in a comprehensive framework for solving the BID while highlighting their usefulness for general CSPs. For example, our query reformulation facilitates a much wider use of relational consistency algorithms than was possible before. In the future, we intend to evaluate these techniques in other application settings. For example, we believe that many resource allocation problems have matching relaxations like we described.

Acknowledgments. Experiments were conducted on the Research Computing Facility at UNL. This research is supported by NSF CAREER Award #0133568 and the Air Force Office of Scientific Research under grant numbers FA9550-04-1-0105 and FA9550-07-1-0416. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any of the above organizations or any person connected with them.

References

1. Michalowski, M., Knoblock, C.: A Constraint Satisfaction Approach to Geospatial Reasoning. In: Proc. of AAAI 2005, pp. 423–429 (2005)
2. Choueiry, B.Y., Iwasaki, Y., McIlraith, S.: Towards a Practical Theory of Reformulation for Reasoning About Physical Systems. *Artificial Intelligence* 162 (1–2), 145–204 (2005)
3. Giunchiglia, F., Walsh, T.: A Theory of Abstraction. *Artificial Intelligence* 57(2-3), 323–389 (1992)
4. Dechter, R., van Beek, P.: Local and global relational consistency. *Journal of Theoretical Computer Science* (1996)
5. Dechter, R.: *Constraint Processing*. Morgan Kaufmann, San Francisco (2003)
6. Russell, S., Norvig, P.: *Artificial Intelligence: A Modern Approach*. In: *Artificial Intelligence: A Modern Approach*, p. 107. Prentice-Hall, Englewood Cliffs (2003)
7. Selman, B., Kautz, H.: Knowledge Compilation and Theory Approximation. *Journal of the ACM* 43(2), 193–224 (1996)

8. Milano, M.(ed.): *Constraint and Integer Programming: Toward a Unified Methodology*. Kluwer Academic Publishers, Dordrecht (2004)
9. Ellman, T.: *Abstraction via Approximate Symmetry*. In: *IJCAI 93*, pp. 916–921 (1993)
10. Régim, J.: *A filtering algorithm for constraints of difference in cps*. In: *AAAI 1994*, pp. 362–367 (1994)
11. Uno, T.: *Algorithms for Enumerating All Perfect, Maximum and Maximal Matchings in Bipartite Graphs*. In: Leong, H.-V., Jain, S., Imai, H. (eds.) *ISAAC 1997*. LNCS, vol. 1350, pp. 92–101. Springer, Heidelberg (1997)
12. Berge, C.: *Graphs and Hypergraphs*. Elsevier, Amsterdam (1973)
13. West, D.: *Introduction to Graph Theory*, 2nd edn. Prentice-Hall, Englewood Cliffs (2001)
14. Hopcroft, J.E., Karp, R.M.: *An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs*. *SIAM* 2, 225–231 (1973)
15. Bessière, C., Meseguer, P., Freuder, E., Larrosa, J.: *On Forward Checking for Non-binary Constraint Satisfaction*. In: Jaffar, J. (ed.) *Principles and Practice of Constraint Programming – CP’99*. LNCS, vol. 1713, pp. 88–102. Springer, Heidelberg (1999)
16. Prosser, P.: *MAC-CBJ: Maintaining Arc Consistency with Conflict-Directed Backjumping*. Technical Report 95/177, Univ. of Strathclyde (1995)
17. Nadel, B.: *Representation Selection for Constraint Satisfaction: A Case Study Using n-Queens*. *IEEE Expert* 5(3), 16–24 (1990)
18. Glaisher, J.: *On the Problem of the Eight Queens*. *Philosophical Magazine* 4(48), 457–467 (1874)
19. Puget, J.: *On the satisfiability of symmetrical constraint satisfaction problems*. In: Komorowski, J., Raś, Z.W. (eds.) *ISMIS 1993*. LNCS, vol. 689, pp. 350–361. Springer, Heidelberg (1993)
20. Holte, R.C., Choueiry, B.Y.: *Abstraction and Reformulation in Artificial Intelligence*. *Philosophical Trans. of the Royal Society Sec. Biological Sciences* 358(1435), 1197–1204 (2003)
21. Razgon, I., O’Sullivan, B., Provan, G.: *Generalizing Global Constraints Based on Network Flows*. In: *Workshop on Constraint Modelling and Reformulation*, pp. 74–87 (2006)

Reformulating Global Constraints: The SLIDE and REGULAR Constraints

Christian Bessiere¹, Emmanuel Hebrard², Brahim Hnich³, Zeynep Kiziltan⁴,
Claude-Guy Quimper⁵, and Toby Walsh⁶

¹ LIRMM, University of Montpellier, France
bessiere@lirmm.fr

² Emmanuel Hebrard, 4C, University College Cork, Ireland
ehebrard@4c.ucc.ie

³ Brahim Hnich, Faculty of Computer Science, Izmir University of Economics, Turkey
brahim.hnich@ieu.edu.tr

⁴ Zeynep Kiziltan, Department of Computer Science, University of Bologna, Italy
zeynep@cs.unibo.it

⁵ Claude-Guy Quimper, Omega Optimisation, Canada
quimper@alumni.uwaterloo.ca

⁶ Toby Walsh NICTA and University of New South Wales, Sydney, Australia
tw@cse.unsw.edu.au

Abstract. Global constraints are useful for modelling and reasoning about real-world combinatorial problems. Unfortunately, developing propagation algorithms to reason about global constraints efficiently and effectively is usually a difficult and complex process. In this paper, we show that reformulation may be helpful in building such propagators. We consider both hard and soft forms of two powerful global constraints, SLIDE and REGULAR. These global constraints are useful to represent a wide range of problems like rostering and scheduling where we have a sequence of decision variables and some constraint that holds along the sequence. We show that the different forms of SLIDE and REGULAR can all be reformulated as each other. We also show that reformulation is an effective method to incorporate such global constraints within an existing constraint toolkit. Finally, this study provides insight into the close relationship between these two important global constraints.

1 Introduction

Global constraints are one of the most important features of constraint programming. Global constraints capture common patterns occurring in models of complex, real-life combinatorial problems. For instance, a common pattern in rostering problems is that sequences of night shifts must be followed by several days off, and that no one is allowed to work more than a certain number of consecutive night shifts. Such patterns can be modelled using a REGULAR constraint [6]. More recently, we have proposed the global SLIDE to model a wide range of patterns appearing in sequencing and other problems [3]. In this paper, we explore the relationship between these two global constraints in depth. We

show that the SLIDE constraint can be efficiently reformulated as the REGULAR constraint and vice versa.

In many real-world problems, it is not possible to find a feasible solution that satisfies all the constraints and preferences of the user. Consider the problem of allocating reviewers for papers submitted to a conference. Typically, a paper must be reviewed by a certain number of reviewers and each reviewer must have a certain number of papers. Reviewers also indicate preferences over the papers that they would be happy to review. Finding an allocation that satisfies the assignment constraints as well as all the reviewer preferences may not be possible. We may however give an additional paper to some reviewers and may assign a paper to a reviewer even if she is not enthusiastic about it, as long as she did not indicate a “conflict of interest”. Soft versions of global constraints are useful to model and solve such over-constrained problems.

A recent direction is to convert over-constrained problems into constraint optimization problems by treating constraint violations as costs. To reason about such problems, we can design specific cost-based propagators. Several such soft global constraints have been proposed to model and solve over-constrained problems effectively and efficiently (e.g., [7,9,10,12]). Whilst efficient propagators have been developed for both the SLIDE and REGULAR constraint and are available in a number of solvers, there has been less work about soft versions of these constraints. For example, no GAC propagators have yet been developed for soft versions of the SLIDE constraint. One of our contributions in this paper is to propose the first such propagators. We show that reformulation is an attractive mechanism also to implement soft versions of the SLIDE and REGULAR global constraints.

This rest of this paper is structured as follows. After giving the necessary formal background in Section 2, we explain in detail in Section 3 the SLIDE and REGULAR constraints. Then we show in Section 4 how SLIDE can be efficiently reformulated as the REGULAR constraint and vice versa. In Section 5, we focus on the soft versions of these global constraints and in Section 6 demonstrate that the different types of SOFTSLIDE constraints can be reformulated as hard forms of the SLIDE or soft form of the REGULAR constraints. We provide experimental proof in Section 7 that reformulating global constraints could be useful. We report related work in Section 8 and conclude in Section 9.

2 Formal Background

A constraint satisfaction problem consists of a set of variables, each with a finite domain of values, and a set of constraints specifying allowed combinations of values for some subset of variables. We consider finite domain integer variables, and use capital letters for variables (e.g. X) and lower case for values (e.g. d). We write $D(X)$ for the domain of a variable X . A constraint C defined on variables $[X_i, \dots, X_j]$ is indicated as $C(X_i, \dots, X_j)$.

Constraint solvers typically explore partial assignments enforcing a local consistency property using either specialised or general purpose algorithms.

A constraint C is *generalised arc consistent* (GAC) iff when a variable is assigned any of the values in its domain, there exist compatible values in the domains of all the other variables of C . For binary constraints, generalised arc consistency is often called simply arc consistency (AC). A constraint C is *bound consistent* (BC) iff when a variable is assigned the maximum (or minimum) value in its domain, there exist compatible values between the maximum and minimum domain value for all the other variables of C .

A deterministic finite automaton (DFA) is described by a 5-tuple $\mathcal{A} = \langle Q, Q_F, q_0, \delta, \Sigma \rangle$ where Σ is an alphabet, Q is a set of states, $q_0 \in Q$ is the initial state, $Q_F \subseteq Q$ is a set of final states, and $\delta \subseteq Q \times \Sigma \times Q$ is a transition table. A sequence s_1, \dots, s_n is accepted by the automaton \mathcal{A} if there exists a sequence of states t_0, \dots, t_n such that $t_0 = q_0$ is the initial state, $t_n \in Q_F$ is a final state, and $\langle t_{i-1}, s_i, t_i \rangle \in \delta$ is a transition in the transition table. A language $\mathcal{L} \subseteq \Sigma^*$ is a (possibly infinite) set of sequences taken from an alphabet Σ . A *regular language* is the set of sequences accepted by a DFA.

The *Hamming distance* between two strings s_1 and s_2 of the same length is the number of positions in which they differ. The Hamming distance is denoted by $H(s_1, s_2)$. For example, the Hamming distance between the strings abc and adc is 1 as they differ only on the second position. The Hamming distance between a string s and a language \mathcal{L} is $\min\{H(s, t) \mid t \in \mathcal{L}\}$. If \mathcal{L} does not contain any string of the same length as s , the distance between s and \mathcal{L} is undefined. The *edit distance* between two strings s_1 and s_2 is the minimum number of character insertions, deletions, and replacements to string s_1 in order to obtain s_2 . The edit distance is denoted by $E(s_1, s_2)$. For example, the edit distance between the strings abc and $aadc$ is 2 as we can replace the character b of the string abc by a and insert a d before c to obtain $aadc$. The edit distance between a string s and a language \mathcal{L} is $\min\{E(s, t) \mid t \in \mathcal{L}\}$. If \mathcal{L} is empty, the distance between s and \mathcal{L} is undefined.

3 SLIDE and REGULAR Constraints

The global REGULAR constraint was introduced by Pesant to model problems in scheduling and rostering [6]. The constraint is specified in terms of a finite automaton which accepts the string of values spelled out by the sequence of variables. More precisely, $\text{REGULAR}(\mathcal{A}, [X_1, \dots, X_n])$ holds iff X_1 to X_n form a string accepted by the DFA \mathcal{A} . Such a global constraint can be used to ensure certain patterns do (or do not) occur over time. For example, in shift rostering, we might have that we cannot work more than three night shifts in a row and once a sequence of night shifts ends, we must have at least two days off. This can easily be specified using a finite automaton. We have a finite automaton \mathcal{A} with the states n_1, n_2, n_3, o_1 and *any*. The transition table is: $\langle \text{any}, \text{day}, \text{any} \rangle$, $\langle \text{any}, \text{night}, n_1 \rangle$, $\langle \text{any}, \text{off}, \text{any} \rangle$, $\langle n_i, \text{night}, n_{i+1} \rangle$, $\langle n_i, \text{off}, o_1 \rangle$, and $\langle o_1, \text{off}, \text{any} \rangle$. For instance, if we are in state *any*, we go to state n_1 with input *night*. Pesant gives a propagator for the REGULAR constraint based on dynamic programming which achieves GAC in $O(nd|Q|)$ time where $|Q|$ is the number of states of the automaton [6].

More recently, we introduced the global SLIDE constraint [3]. We begin with its simplest form. If C is a constraint of arity k then $\text{SLIDE}(C, [X_1, \dots, X_n])$ holds iff $C(X_i, \dots, X_{i+k-1})$ itself holds for $1 \leq i \leq n - k + 1$. That is, we slide the constraint C down the sequence of variables, X_1 to X_n . For example, consider the car sequencing problem (prob001 in CSPLib) where we need to decide the order in which to build cars on an assembly line. We might want to ensure that no more than one out of every two cars has the sun roof option as it takes extra time to fit a sun roof. This can easily be specified with a SLIDE constraint. We slide a binary constraint down a sequence of decision variables representing the order in which cars will be produced. This binary constraint ensures that one of or both of the variables within its scope does not represent a car with the sun roof option. A variation of the dual encoding can be used to maintain GAC on such a SLIDE constraint in $O(nkd^k)$ time [3].

A more complex form of the SLIDE constraint permits us to slide down two or more sequences of variables at the same time. For instance, if C is a constraint of arity $2k$ then $\text{SLIDE}(C, [X_1, \dots, X_n], [Y_1, \dots, Y_n])$ holds iff $C(X_i, \dots, X_{i+k-1}, Y_i, \dots, Y_{i+k-1},)$ itself holds for $1 \leq i \leq n - k + 1$. That is, we slide the constraint C down the two sequences of variables. Consider, for example, the global contiguity constraint [5]. This ensures that within a sequence of 0/1 variables, X_1 to X_n , the 1's occur in a continuous block. We can model this by a SLIDE constraint down two sequences of 0/1 variables, X_1 to X_n and Y_1 to Y_n . The second sequence of variables, Y_1 to Y_n record if we have met the block of 1's yet or not. The constraint being slid, $C(X_i, X_{i+1}, Y_i, Y_{i+1})$ holds iff $(Y_i = 0, X_i = 0, X_{i+1} = Y_{i+1})$ or $(Y_i = Y_{i+1} = 1 \text{ and } X_i \geq X_{i+1})$. Such more complex forms of SLIDE can be reformulated as the simple form of SLIDE down a single sequence. We merely need to interleave the different sequences and then slide a suitably modified constraint over these interleaved sequences. (See [3] for details.)

4 Reformulating SLIDE and REGULAR

We first show that SLIDE and REGULAR can be reformulated as each other. As well as providing insight into the relationship between the two global constraints, these reformulations will be useful in providing propagators for soft versions of these global constraints.

4.1 REGULAR as SLIDE

In [8], we give a simple reformulation of the REGULAR constraint in terms of a SLIDE constraint. In addition to the sequence of variables along which the REGULAR constraint is defined, we introduce a sequence of variables for the state of the automaton. We then slide the transition relation down the two sequences of variables. For instance, consider again the shift rostering problem from the last section. We can reformulate $\text{REGULAR}(\mathcal{A}, [X_1, \dots, X_n])$ as $\text{SLIDE}(C, [X_1, \dots, X_{n+1}], [Q_1, \dots, Q_n])$ where X_{n+1} is a dummy variable, Q_i are variables representing the state of the automaton, and $C(X_i, X_{i+1}, Q_i, Q_{i+1})$ holds iff we move from state Q_i to Q_{i+1} on

seeing X_i . Observe that X_{i+1} is not used in the definition of C . We let it in its scope just to be consistent with the simplified presentation of SLIDE on multiple sequences that we presented in Section 3.

Enforcing GAC on this reformulation achieves GAC on the original REGULAR constraint. Hence, reformulation does not hinder propagation. This reformulation is also optimal in the sense that, we can enforce GAC on either the REGULAR constraint or its reformulation into SLIDE in $O(nd|Q|)$ time. This complexity is lower than the complexity of SLIDE in general because of the characteristics of the constraint being slid (see [3] for details). As we show in the experimental section, this reformulation is also a practical means to propagate the REGULAR constraint. By reformulating REGULAR as a SLIDE constraint, we get an efficient and incremental propagator that can outperform Pesant's propagator for REGULAR based on dynamic programming.

4.2 SLIDE as REGULAR

The reverse is also possible. That is, we can reformulate any instance of the SLIDE constraint using a REGULAR constraint. Again this is optimal in the sense that we can enforce GAC on either the SLIDE constraint or the reformulation into REGULAR in $O(nkd^k)$ time. We prove this claim by constructing a DFA \mathcal{A} recognizing the language accepted by a SLIDE constraint.

Let $\Sigma = \bigcup_{i=1}^n D(X_i)$ be the alphabet and k be the arity of the constraint C . The states of \mathcal{A} are given by the set of sequences $Q = \bigcup_{i=0}^{k-1} \Sigma^i$. The empty sequence $\epsilon \in Q$ is the initial state. Any sequence of length $k - 1$ is a final state. Let $T = \{[s_1, \dots, s_k] \mid C([s_1, \dots, s_k])\}$ be the set of sequences accepted by the constraint C . We construct the transition table δ of \mathcal{A} as follows. Let w be a sequence of length strictly smaller than $k - 1$ and $c \in \Sigma$ be a character from the alphabet. Let wc be the concatenation of the sequence w and the character c . We have a transition $\langle w, c, wc \rangle \in \delta$ if there exists a sequence in T starting with wc . Let $a, b \in \Sigma$ be two characters and $w \in \Sigma^{k-2}$ be a sequence of length $k - 2$ such that awb is a sequence in T . Then we have the transition $\langle aw, b, wb \rangle \in \delta$. Notice that a state w can only be visited after parsing the sub-sequence w .

Example 1. Consider the alphabet $\Sigma = \{a, b\}$ and the constraint C that accepts any sequence of length three but the sequences aaa and bbb . We obtain the DFA depicted in Figure 11.

The DFA \mathcal{A} constructed in this way represents the SLIDE constraint.

Theorem 1. *If n is greater than or equal to the arity of C , then the language $\{X_1 \dots X_n \mid \text{SLIDE}(C, [X_1, \dots, X_n])\}$ formed by the sequences satisfying the SLIDE constraint is equal to the set of sequences of length n recognized by the DFA \mathcal{A} .*

Proof. We first prove that every sequence s of length n accepted by \mathcal{A} is also accepted by the SLIDE constraint. Let w be the first $k - 1$ characters in s . By construction of \mathcal{A} after reading these $k - 1$ characters, the current state is w and

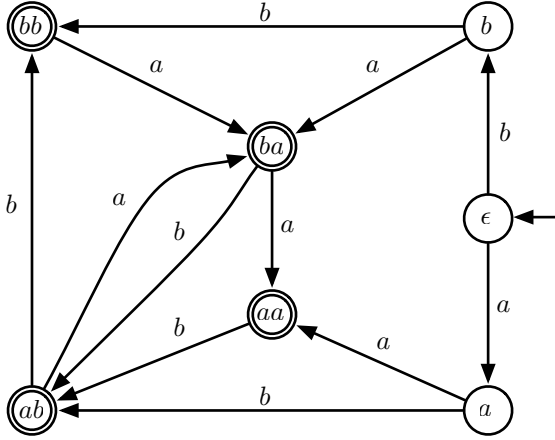


Fig. 1. DFA corresponding to Example III

there exists a sequence in T starting with w . Assume that the $k - 1$ characters following a position i in s are given by aw where a is a character and w is a sequence of length $k - 2$. Also assume that there exists a sequence in T starting with aw and that after reading aw , \mathcal{A} is in state aw . By construction of \mathcal{A} when reading the character b at position i , the state of \mathcal{A} changes from aw to wb . The transition guarantees that the sequence awb belongs to T and that the constraint C is satisfied at position i . By inductively repeating the argument, we conclude that the constraint C is satisfied at every position and that consequently, the SLIDE constraint is satisfied.

We now prove that if the sequence s satisfies the SLIDE constraint, then it is accepted by \mathcal{A} . Let awb be the first k characters of s where a and b are two characters and w is a sequence of $k - 2$ characters. Since the SLIDE constraint is satisfied, the sequence awb satisfies the constraint C and belongs to the set T . Therefore, there is a series of states q_0, \dots, q_{k-1} that parses the sequence aw where state q_i is the first i characters of aw . Suppose that the i^{th} character to be parsed is b , that the last $k - 1$ parsed characters are the character a followed by the sequence w , and that \mathcal{A} is in state aw . Since the SLIDE constraint is satisfied, the sequence awb satisfies C . Therefore, there exists a transition $\langle aw, b, wb \rangle \in \delta$ that parses the character b and leads to the state wb . Notice that wb are the last $k - 1$ parsed characters. By inductively repeating the argument, we conclude that there exists a parsing for any sequence satisfying the slide constraint. Consequently, the sequence s is accepted by \mathcal{A} . \square

5 Softening SLIDE and REGULAR

As we discussed in the introduction, real world problems are often over-constrained. One mechanism to deal with such over-constrained problems is to introduce a cost

function, and find a solution of minimal cost from a feasible solution. We will formalize this process as follows. Let $C(X_1, \dots, X_n)$ be a hard global constraint like SLIDE or REGULAR. Given a distance function d between strings, the soft constraint $C_{soft}(X_1, \dots, X_n, Z)$ holds iff $Z = \min\{d(a_1, \dots, a_n, b_1, \dots, b_m) \mid a_i \in D(X_i) \forall i \in 1..n \ \& \ C(b_1, \dots, b_m)\}$. Distance might, for instance, be Hamming (in which case $n = m$) or edit distance. As stated in Section 2, d is not defined if the set of strings satisfying C is empty.

As an example, the constraint $\text{SOFTSLIDE}_H(C, [X_1, \dots, X_n], D)$ holds iff D is the Hamming-distance between the sequence $[X_1, \dots, X_n]$ and the language accepted by the SLIDE constraint. Similarly, the constraint $\text{SOFTSLIDE}_E(C, [X_1, \dots, X_n], D)$ holds iff D is the edit-distance between the sequence $[X_1, \dots, X_n]$ and the language accepted by the SLIDE constraint. Similarly, SOFTREGULAR_H and SOFTREGULAR_E are soft forms of REGULAR obtained by applying Hamming and edit distance based violations to REGULAR respectively [10]. For example, $\text{SOFTREGULAR}_E(\mathcal{A}, [X_1, \dots, X_n], D)$ is satisfied iff D is the edit-distance between the sequence $[X_1, \dots, X_n]$ and the language accepted by \mathcal{A} . In [10], the propagation algorithm for REGULAR based on dynamic programming is modified in two different ways to maintain GAC on SOFTREGULAR_H and SOFTREGULAR_E , respectively.

6 SOFTSLIDE Constraint

We will show that the different types of SOFTSLIDE constraints can be reformulated as hard forms of the SLIDE or soft form of the REGULAR constraints. Reformulation thus provides a simple mechanism to propagate the soft forms of these global constraints.

6.1 SOFTSLIDE_H as SLIDE

Let us consider a $\text{SOFTSLIDE}_H(C, [X_1, \dots, X_n], D)$ where the arity of constraint C is k and where $n \geq k$. In order to reformulate this SOFTSLIDE constraint as a SLIDE constraint, we introduce two sequences of extra variables.

The first sequence contains $n+k-1$ variables $[S_1, \dots, S_{n+k-1}]$ that we build in such a way that S_1, \dots, S_n can be any string accepted by $\text{SLIDE}(C, [S_1, \dots, S_n])$. The domain of each S_i contains all the values that appear in at least one tuple belonging to C . The variables S_{n+1} to S_{n+k-1} must not be forced to satisfy the constraint being slid. Hence, a dummy value '*' is added to the domain of every $S_i, i > n$. Furthermore, C' is defined as a relaxation of C that contains a tuple t iff t belongs to C or t contains at least one dummy value '*'. For instance, if C is a ternary constraint allowing the following set of tuples $\{\langle a, b, a \rangle, \langle b, c, a \rangle, \langle a, b, c \rangle\}$, then the domain of each $S_i, i \leq n$, is $\{a, b, c\}$, the domain of each $S_i, i > n$, is $\{a, b, c, *\}$, and $C' = C \cup \{\langle d_1, d_2, d_3 \rangle \mid \exists i \in 1..3, d_i = *\}$.

The second sequence of variables contains $n+1$ variables $[D_1, \dots, D_{n+1}]$ which provide the cumulative count of the number of discrepancies with respect to the previous sequence. Then we introduce the following $k+3$ -ary constraint:

$$C_H(X_i, S_i, \dots, S_{i+k-1}, D_i, D_{i+1}) \Leftrightarrow C'(S_i, \dots, S_{i+k-1}) \wedge D_{i+1} = D_i + (S_i \neq X_i)$$

This constraint ensures that C' is satisfied by the sequence $[S_i, \dots, S_{i+k-1}]$ (that is, $[S_i, \dots, S_{i+k-1}]$ satisfies C if $i+k-1 \leq n$), and $D_{i+1} = D_i$ if $S_i = X_i$, otherwise $D_{i+1} = D_i + 1$. Using the more complex form of SLIDE over multiple sequences, we can thus reformulate the SOFTSLIDE_H constraint by sliding C_H over the three sequences $[S_1, \dots, S_{n+k-1}]$, $[X_1, \dots, X_n]$, and $[D_1, \dots, D_{n+1}]$ and by constraining D_1 to be 0 and D_{n+1} to be equal to D . That is, we have:

$$\text{SOFTSLIDE}_H(C, [X_1, \dots, X_n], D)$$

\Leftrightarrow

$$\text{SLIDE}(C_H, [S_1, \dots, S_{n+k-1}], [X_1, \dots, X_n], [0, D_2, \dots, D_n, D])$$

Enforcing GAC on SLIDE is in $O(nkd^k)$, where n is the length of the sequence and k the arity of the constraint being slid. In the case of the reformulation of SOFTSLIDE_H as SLIDE , the constraint to be slid has arity $k+3$. Thus, the time complexity of enforcing GAC on SOFTSLIDE_H using this reformulation is in $O(nkd^{k+3})$ where d is the number of values that are used in tuples allowed by the constraint C .

Note that for this encoding to work correctly, the variables $[S_1, \dots, S_{n+k-1}]$ should not be constrained by other constraints, so that GAC on $\text{SLIDE}(C, [S_1, \dots, S_{n+k-1}])$ guarantees a solution. Since such variables are introduced during the reformulation, they will be invisible to the users of the SOFTSLIDE_H constraint, hence such an assumption is reasonable.

6.2 SOFTSLIDE_E as SOFTREGULAR_E

By using the same reformulation we proposed of SLIDE as REGULAR , we can reformulate SOFTSLIDE_E as SOFTREGULAR_E . The set of strings accepted by the hard version of the SOFTSLIDE_E constraint must not be empty because the propagator of SOFTREGULAR_E described in [10] is defined for automata accepting a non empty language. This propagator achieves GAC on the variables X_i for $1 \leq i \leq n$ and BC on D in $O(n|\delta| + n|Q| \log(n|Q|))$ steps. The DFA \mathcal{A} has $O(|\Sigma|^i)$ states labelled with a sequence of length i for a total of $|Q| = \sum_{i=0}^{k-1} O(|\Sigma|^i) = O(|\Sigma|^k)$ states. The outgoing degree of every state is bounded by $|\Sigma|$. We therefore have $|\delta| = O(|\Sigma|^{k+1})$ transitions. Filtering the SOFTSLIDE_E constraint therefore requires $O(n|\Sigma|^{k+1} + n|\Sigma|^k \log(n|\Sigma|^k)) = O(n|\Sigma|^{k+1} + nk|\Sigma|^k \log(n|\Sigma|))$ time.

7 Experimental Analysis

We now show that reformulation is an effective mechanism to provide propagators for SLIDE and REGULAR constraints. First, we compare reasoning about a REGULAR constraint encoded as SLIDE , with reasoning about it directly using Pesant's GAC propagator [6]. Then, we analyse the reverse reformulation and compare reasoning a SLIDE constraint reformulated as a REGULAR , with reasoning about it directly using the GAC propagator given in [3]. Pesant presents

two propagators. We implemented the one that keeps track of all supports for each value in the domain of every variable. Whilst the first set of experiments are done using ILOG Solver 6.1 on a 900 MHz Pentium running Linux Debian, the second set is done using ILOG Solver 6.2 on a 2.8GHz Intel Xeon computer running Linux FC2.

7.1 REGULAR as SLIDE

As in [6], we generated random automata with $|Q|$ states and an alphabet of size $|\Sigma|$. We selected 30% of all possible tuples $(c, q_i) \in \Sigma \times Q$ and randomly chose a state $q_j \in Q$ to form the transition $T(c, q_i) = q_j$. We obtained the set of final states F by randomly selecting 50% of the states in Q . Following Pesant, we used a random variable ordering and random value selection. All experiments are averaged over 30 runs. Table 1 shows the results. We observe that the reformulation of REGULAR constraints in terms of SLIDE is as efficient as and most of the times slightly more efficient than propagating directly the REGULAR constraints. The propagator for SLIDE uses a sequence of built-in TABLE constraints. We conjecture that these are highly optimized and contribute to the performance offered by SLIDE.

We also ran experiments on a model for the Mystery Shopper problem due to Helmut Simonis that appears in CSPLib (prob004). This model contains a large number of AMONG constraints. We represented these AMONG constraints using REGULAR constraints, and again either reasoned with these REGULAR constraints directly using Pesant's propagator or reformulated them using SLIDE constraints.

Results are given in Table 2. All instances solved in the experiments use a time limit of 5 minutes. Both methods achieve GAC on the AMONG constraint, so the search trees are identical and it is only the efficiency of the propagator which differ. Again, reformulation of REGULAR using SLIDE is slightly more efficient.

7.2 SLIDE as REGULAR

We consider a variant of the Nurse Scheduling Problem [4] that consists of generating a schedule for each nurse of shifts duties and days off within a short-term planning period. There are three types of shifts (day, evening, and night). We ensure that (1) each nurse should take a day off or be assigned to an available shift; (2) each shift has a minimum required number of nurses; (3) each nurse work load should be between specific lower and upper bounds; (4) each nurse can work at most 5 consecutive days; (5) each nurse must have at least 12 hours of break between two shifts; (6) the shift assigned to a nurse cannot change more than once every three days. We develop two models to solve this problem. In both, we introduce one variable for each nurse and each day, indicating to what type of shift, if any, this nurse is affected on this day. The constraints (1)-(3) are enforced using a set of global cardinality constraints. The constraints (4), (5) and (6) form sequences of respectively 6-ary, binary and ternary constraints. Notice that (4) is monotone, hence we simply post these constraints in both

Table 1. Time in seconds to find a sequence satisfying a randomly generated automaton either using Pesant’s propagator for the REGULAR constraint or reformulating it as a SLIDE constraint

n	$ \Sigma $	$ Q $	REGULAR	REGULAR as SLIDE
25	5	10	0.0032	0.0031
		20	0.0029	0.0025
		40	0.0052	0.0046
		80	0.0079	0.0041
25	10	10	0.0053	0.0038
		20	0.0099	0.0063
		40	0.0165	0.0087
		80	0.0284	0.0136
25	20	10	0.0113	0.0057
		20	0.0195	0.0083
		40	0.0399	0.0140
		80	0.0812	0.0226
n	$ \Sigma $	$ Q $	REGULAR	REGULAR as SLIDE
50	5	10	0.0047	0.0051
		20	0.0047	0.0037
		40	0.0101	0.0086
		80	0.0168	0.0087
50	10	10	0.0105	0.0071
		20	0.0207	0.0129
		40	0.0359	0.0185
		80	0.0631	0.0301
50	20	10	0.0232	0.0119
		20	0.0396	0.0177
		40	0.0814	0.0289
		80	0.1655	0.0457

models. The conjunction of constraints (5) and (6) is slid using the tuple encoding of SLIDE in the first model, and an encoding of SLIDE using REGULAR in the second model.

We test the models by using the instances available at <http://www.projectmanagement.ugent.be/nsp.php> in which nurses have no maximum workload, but a set of preferences is to be optimised. We ignore these preferences and post a constraint for bounding the maximum workload to at most 5 day shifts, 4 evening shifts and 2 night shifts per nurses and per week. Similarly, each nurse must have at least 2 rest days per week. We solve a sample of 99 instances involving a crew of 30 nurses to schedule over 28 days. We use the same static variable ordering for both models. The days are scheduled in chronological order, and within each day, we allocate a shift to every nurse in lex order. Initial experiments show that this simple heuristic is more efficient than dynamic minimum domain heuristic.

In Table 3, we report the mean fails and cpu time required to solve the instances. We observe that the first model is about 15% faster than the second model.

Table 2. Mystery Shopper problem, REGULAR v. REGULAR as SLIDE. #fails and cpu time are only averaged on instances solved by both methods.

Size	REGULAR			REGULAR as SLIDE		
	#fails	cpu time	#solved	#fails	cpu time	#solved
10	6	0.01022	9/10	6	0.00755	9/10
15	8342	1.19897	32/52	8342	1.15954	32/52
20	12960	5.63347	21/35	12960	3.40063	21/35
25	6186	1.41279	4/20	6186	0.87862	4/20
30	1438	0.72189	3/10	1438	0.47626	3/10
35	6297	3.73623	20/56	6297	2.36849	20/56

Table 3. Nurse scheduling problem (30 nurses, 28 days), SLIDE v. SLIDE as REGULAR. #fails and cpu time are only averaged on instances solved by both methods.

	SLIDE	SLIDE as REGULAR
instances solved	56/99	56/99
time	3.77	4.39
backtracks	4761	4761

8 Related Work

Reformulating new global constraints in terms of those that already available within the constraint toolkit has started to gain attention within the constraint programming community. For instance, in [11], the AMONGSEQ constraint used in car sequencing on a production line is studied and alternative propagation methods are discussed. One approach reformulates AMONGSEQ as a REGULAR constraint. This reformulation is shown to be the most efficient in practice compared to the other proposed propagators.

Given the large number of global constraints that have been identified, another direction of study is “general-purpose” global constraints. Such constraints can be used in conjunction with the primitive constraints to reformulate a wide range of global constraints without the need to extend the constraint toolkits. This is especially useful if a constraint toolkit does not provide a propagator for the global constraint or if the constraint is difficult to propagate. SLIDE is such a general constraint because it helps encode and propagate many sequencing constraints [3]. Other examples of general constraints are RANGE and ROOTS [2]. They are shown to be very useful for reformulating diverse global constraints appearing in counting and occurrence problems.

One of the simplest ways to soften the SLIDE constraint is to relax the number of times the slid constraint holds on the sequence. This gives the CARDPATH constraint. CARDPATH($C, [X_1, \dots, X_n], N$) holds iff C holds N times on the sequence $[X_1, \dots, X_n]$ [1]. Interestingly, the CARDPATH constraint can itself be reformulated as a SLIDE constraint [3]. We can therefore use the propagator for SLIDE to propagate the CARDPATH constraint. In fact, this reformulation is the first and only method proposed so far in the literature for enforcing GAC on CARDPATH.

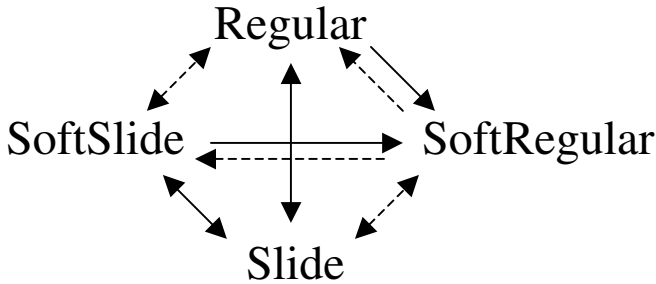


Fig. 2. The relationship between SLIDE, SOFTSLIDE, REGULAR, and SOFTREGULAR constraints

9 Conclusions

To model real-world constraint problems and to solve them efficiently, many global constraints have been proposed in recent years. In this paper, we have focused on two important global constraints, SLIDE and REGULAR which are useful for encoding and propagating a wide range of rostering and sequencing problems. Since problems are often over-constrained, we have also studied soft forms of these global constraints. We showed that the different forms of SLIDE and REGULAR can all be reformulated as each other. We also showed that reformulation is an effective method to incorporate such global constraints within an existing constraint toolkit. This study has provided insight into the close relationship between these two important global constraints.

The relationships depicted in Figure 2 demonstrate the close links between the hard and soft versions of the SLIDE and REGULAR constraints. An arrow from a constraint C_i to a constraint C_j indicates in the figure that C_i can be reformulated as C_j . A thick arrow is either due to findings in this paper or due to the fact that a soft form of a constraint can be used to propagate its hard form by not allowing any violation. The dashed arrows can be obtained by transitivity from the thick arrows. For instance, given that REGULAR can be reformulated as SLIDE which can itself be reformulated as SOFTSLIDE, we can derive that REGULAR can be reformulated as SOFTSLIDE.

References

1. Beldiceanu, N., Carlsson, M.: Revisiting the cardinality operator and introducing cardinality-path constraint family. In: Codognet, P. (ed.) ICLP 2001. LNCS, vol. 2237, pp. 59–73. Springer, Heidelberg (2001)
2. Bessière, C., Hebrard, E., Hnich, B., Kiziltan, Z., Walsh, T.: The range and roots constraints: Specifying counting and occurrence problems. In: Proc. of IJCAI'05, pp. 60–65. Professional Book Center (2005)
3. C. Bessière, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. The Slide Meta-Constraint. Comic technical report, 2006 (available at <http://homes.ieu.edu.tr/bhnych/comic/>)

4. Burke, E.K., Causmaecker, P.D., Berghe, G.V., Landeghem, H.V.: The state of the art of nurse rostering. *Journal of Scheduling* 7(6), 441–499 (2004)
5. Maher, M.: Analysis of a global contiguity constraint. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, Springer, Heidelberg (2002)
6. Pesant, G.: A regular language membership constraint for finite sequences of variables. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 482–295. Springer, Heidelberg (2004)
7. Petit, T., Régim, J-C., Bessière, C.: Specific filtering algorithms for over-constrained problems. In: Drira, K., Martelli, A., Villemur, T. (eds.) Cooperative Environments for Distributed Systems Engineering. LNCS, vol. 2236, pp. 451–463. Springer, Heidelberg (2001)
8. Quimper, C.-G., Walsh, T.: Global grammar constraints. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 751–755. Springer, Heidelberg (2006)
9. van Hoes, W-J.: A hyper-arc consistency algorithm for the soft alldifferent constraint. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 679–689. Springer, Heidelberg (2004)
10. van Hoes, W-J., Pesant, G., Rousseau, L-M.: On global warming: Flow-based soft global constraints. *Journal of Heuristics* 12(4-5), 347–373 (2006)
11. van Hoes, W-J., Pesant, G., Rousseau, L-M., Sabharwal, A.: Revisiting the sequence constraint. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 620–634. Springer, Heidelberg (2006)
12. Zanarini, A., Milano, M., Pesant, G.: Improved algorithm for the soft global cardinality constraint. In: Beck, J.C., Smith, B.M. (eds.) CPAIOR 2006. LNCS, vol. 3990, pp. 288–299. Springer, Heidelberg (2006)

Relaxation of Qualitative Constraint Networks

Dominique D’Almeida, Jean-François Condotta,
Christophe Lecoutre, and Lakhdar Saïs

CRIL-CNRS, Université d’Artois, rue de l’Université, 62307 Lens, France
{dalmeida,condotta,lecoutre,sais}@cril.univ-artois.fr

Abstract. In this paper, we propose to study the interest of relaxing qualitative constraints networks by using the formalism of discrete Constraint Satisfaction Problem (CSP). This allows us to avoid the introduction of new definitions and properties in the domain of qualitative reasoning. We first propose a general (but incomplete) approach to show the unsatisfiability of qualitative networks, by using a relaxation on any set of relations. Interestingly enough, for some qualitative calculi, the proposed scheme can be extended to determine the satisfiability of any qualitative network, leading to an original, simple and complete method. However, as the efficiency of our approach depends on the chosen relaxation, total relations should be preferred due to their connections with the hardness of constraint networks. We then present some preliminary experimental results, with respect to unsatisfiability, which show some promising improvements on some classes of random qualitative networks.

1 Introduction

The need for reasoning about time and space arises in many areas of Artificial Intelligence, including computer vision, natural language understanding, geographic information systems (GIS), scheduling, planning, diagnosis and genetics. Numerous formalisms for representing and reasoning about time and space in a qualitative way have been proposed in the past two decades [1,28,22,3,27,19,4].

Those formalisms involve a finite set of basic relations denoting qualitative relationships between temporal or spatial entities. Intersection, overlapping, containment, precedence are examples of such qualitative relationships. For instance, in the field of qualitative reasoning about temporal data, there is a well-known formalism called Allen’s calculus [1]. It is based on intervals of the rational line for representing temporal entities and thirteen basic relations between such intervals are used to represent the qualitative situations between temporal entities: an interval can follow another one, meet another one, and so on.

Typically, Qualitative Constraint Networks (QCNs) are used to express information on a spatial or temporal situation. Each constraint of a QCN represents a set of acceptable qualitative configurations between some temporal or spatial entities and is defined by a set of basic relations. The total relation, which is the set of all basic relations, is the term used to describe a total uncertainty in the configurations. The density of such relations in a qualitative networks can take part to the difficulty to solve those problems.

The aim of this paper is to demonstrate that a relaxation of some relations can lead, in some cases, to a significant computational speed up of the satisfiability checking task. To avoid the introduction of new particular definitions (and properties) in qualitative reasoning, when relaxation is applied, a discrete encoding preserving the chosen relations of qualitative constraint networks is used. Using a relaxation on any set of relations, it is then possible to show the unsatisfiability of some networks. Interestingly enough, for some qualitative calculi, this basic relaxation scheme is extended to determine the satisfiability of any qualitative constraint network leading to an original, simple and complete method. Nevertheless, even if the approach can be used in the general case, we focus our attention on relaxing total relations since these relations are intuitively related to computational efficiency.

Relaxation based approaches are widely used in many domains ranging from constraint satisfaction to linear programming and knowledge representation. For example, in constraint satisfaction, relaxations are used to solve dynamic [20], over-constrained [21] and distributed [31] constraint networks. Some other works are based on concepts of abstract interpretation [14,23], theory of abstraction [16] or theory approximation [8,29]. Whereas in [9], abstract interpretation [14,23] is exploited to improve constraint solving in an object-oriented context, the concept of Galois insertion (at the heart of abstract interpretation) has also been used to deal with flexible constraints [5,6]. Generally speaking, many abstractions (e.g. [15,10,30,11]) proposed in the literature can be seen as a kind of value or variable clustering. The framework introduced in [24] allows to deal with general clustering which means that an element (a value or a variable) can belong to several clusters.

The paper is organized as follows. In the next section, some background on qualitative and discrete formalisms is provided. Then we present the general relaxation scheme that we propose. Before concluding, some experimental results on some classes of random qualitative constraint networks are presented.

2 Technical Background

In this section, we provide the technical background useful for the reading of this paper. First, we present the concept of qualitative calculus before introducing qualitative constraint networks. Then, we introduce discrete constraint networks and describe an encoding of qualitative constraint networks into discrete ones.

2.1 Qualitative Calculus

A qualitative calculus involves a finite set B of binary¹ relations, called basic relations, defined on a domain D . The elements of D represent temporal or spatial entities. Each basic relation of B corresponds to a particular possible configuration between two temporal or spatial entities. The relations of B

¹ In this paper, we focus on binary relations but this work can be extended to non-binary ones.

are jointly exhaustive and pairwise disjoint, which means that any pair of elements of D belongs to exactly one basic relation in B . Moreover, for each basic relation $B \in B$ there exists another basic relation of B , denoted by B^\sim , corresponding to the transposition of B . In addition, we suppose that a particular relation of B , denoted by ld , is the identity relation on D . The set A is defined as the set of relations corresponding to all possible unions of the basic relations: $A = \{\cup E : E \subseteq B\}$. It is customary to represent an element $B_1 \cup \dots \cup B_m$ (with $B_i \in B$ for each i such that $1 \leq i \leq m$) of A by the set $\{B_1, \dots, B_m\}$ belonging to 2^B . Hence, we make no distinction between A and 2^B in the rest of this paper.

As an example, consider the well-known temporal qualitative formalism called Allen’s calculus [2]. It uses intervals of the rational line for representing temporal entities. Hence, D is the set $\{(x^-, x^+) \in \mathbb{Q} \times \mathbb{Q} : x^- < x^+\}$. The set of basic relations consists of a set of thirteen binary relations $B = \{eq, b, bi, m, mi, o, oi, s, si, d, di, f, fi\}$ corresponding to all possible configurations between two intervals. These basic relations are depicted in Figure 1. We have $ld = eq$.

Relation	Symbol	Inverse	Meaning
precedes	b	bi	
meets	m	mi	
overlaps	o	oi	
starts	s	si	
during	d	di	
finishes	f	fi	
equals	eq	eq	

Fig. 1. The basic relations of Allen’s calculus

As a set of subsets, A is equipped with the usual set-theoretic operations including intersection (\cap) and union (\cup). As a set of binary relations, it is also equipped with the operation of converse (\sim) and an operation of composition (\circ) sometimes called weak composition or qualitative composition. The converse of a relation R in A is the union of the transpositions of the basic relations contained in R . The composition $A \circ B$ of two basic relations A and B is the relation $R = \{C \in B \mid \exists x, y, z \in D^3, x A y, y B z \text{ and } x C z\}$. The composition $R \circ S$ of $R, S \in A$ is the relation $T = \bigcup_{A \in R, B \in S} \{A \circ B\}$. Computing the results of these various operations for relations of 2^B can be done efficiently by using tables giving the results of these operations for the basic relations of B . For instance, consider the relations $R = \{eq, b, o, si\}$ and $S = \{d, f, s\}$ of Allen’s calculus, we have $R^\sim = \{eq, bi, oi, s\}$. The relation $R \circ S$ is $\{d, f, s, b, o, m, eq, si, oi\}$.

2.2 Qualitative Constraint Networks

A qualitative constraint network (QCN) is a pair composed of a set of variables and a set of constraints. Each variable represents a spatial or temporal entity of the problem that is represented, and each constraint consists of a set of acceptable basic relations (the possible configurations) between two variables. More formally, a QCN is defined in the following way:

Definition 1. A QCN \mathcal{N} is a pair (V, Q) where $V = \{v_1, \dots, v_n\}$ is a finite set of n variables and Q is a map that assigns to each pair (v_i, v_j) of $V \times V$ a set $Q(v_i, v_j) \in 2^{\mathbf{B}}$ of basic relations. $Q(v_i, v_j)$ will also be denoted by Q_{ij} . Q is such that $Q_{ii} \subseteq \{\text{Id}\}$ and $Q_{ij} = Q_{ji}^{\sim}$ for all $v_i, v_j \in V$.

A solution of a QCN \mathcal{N} is a map σ from V to \mathbf{D} such that $(\sigma(v_i), \sigma(v_j))$ satisfies Q_{ij} for all $v_i, v_j \in V$. \mathcal{N} is consistent iff it admits a solution. A QCN $\mathcal{N}' = (V', Q')$ is a sub-QCN of \mathcal{N} (denoted by $\mathcal{N}' \subseteq \mathcal{N}$) if and only if $V = V'$ and $Q'_{ij} \subseteq Q_{ij}$ for all $v_i, v_j \in V$. A QCN $\mathcal{N}' = (V', Q')$ is equivalent to \mathcal{N} if and only if $V = V'$ and both networks have the same solutions. \mathcal{N} is atomic iff each constraint of \mathcal{N} contains exactly one basic relation. A scenario of \mathcal{N} is an atomic sub-QCN of \mathcal{N} . We will denote by $\text{scen}(\mathcal{N})$ and $\text{sol}(\mathcal{N})$ a scenario and a solution of \mathcal{N} , respectively.

Given a QCN \mathcal{N} , the main issue to be addressed is the consistency problem: to decide whether or not \mathcal{N} admits (at least) a solution. Most of the algorithms used for solving this problem are based on a method that we call the \circ -closure method, also called weak composition closure and denoted WC . The \circ -closure method is a constraint propagation method allowing to enforce the $(0, 3)$ -consistency of \mathcal{N} , which means that all restrictions of \mathcal{N} to 3-variables are consistent. The \circ -closure method involves iteratively performing the following operation: $Q_{ij} := Q_{ij} \cap (Q_{ik} \circ Q_{kj})$, for all v_i, v_j, v_k of V , until a fix-point is reached. This method yields a sub-QCN $\mathcal{N}' = (V, Q')$ of \mathcal{N} which is equivalent to it, and such that $Q'_{ij} \subseteq Q'_{ik} \circ Q'_{kj}$, for all v_i, v_j, v_k of V . This last condition is expressed by saying that the sub-network is \circ -closed (to simplify, we will assume that a \circ -closed QCN does not contain the empty relation associated with a constraint).

2.3 Discrete Constraints Networks

Definition 2. A Discrete Constraint Network (DCN) \mathcal{P} is a pair (X, C) where X is a finite set of variables and C a finite set of constraints. Each variable $x \in X$ has an associated domain, denoted $\text{dom}^{\mathcal{P}}(x)$, which represents the set of values allowed for x . Each constraint $c \in C$ involves a subset of variables of X , called scope and denoted $\text{scp}(c)$, and has an associated relation denoted $\text{rel}^{\mathcal{P}}(c)$, which represents the set of tuples allowed for the variables of its scope.

When possible, we will write $\text{dom}(x)$ and $\text{rel}(c)$ instead of $\text{dom}^{\mathcal{P}}(x)$ and $\text{rel}^{\mathcal{P}}(c)$. If \mathcal{P} and \mathcal{P}' are two DCNs defined on the same sets of variables X and constraints C , then we will write $\mathcal{P} \preceq \mathcal{P}'$ (and we will say that \mathcal{P} is a subnetwork of \mathcal{P}') iff $\forall x \in X, \text{dom}^{\mathcal{P}}(x) \subseteq \text{dom}^{\mathcal{P}'}(x)$. A solution to a discrete constraint network is an assignment of values to all the variables such that all the constraints are

satisfied. A constraint network is said to be satisfiable or consistent iff it admits at least one solution. The Constraint Satisfaction Problem (CSP) is the NP-complete task of determining whether a given constraint network is satisfiable. A CSP instance is then defined by a constraint network, and solving it involves either finding one (or more) solution or determining its unsatisfiability.

To solve a CSP instance, one can apply inference or search methods. Usually, domains of variables are reduced by removing inconsistent values, i.e. values that can not occur in any solution. Indeed, it is possible to filter domains by considering some properties of constraint networks. Generalized Arc Consistency (GAC) remains the central property of constraint networks and establishing GAC on a given network \mathcal{P} involves removing all values that are not generalized arc-consistent. Remark that for binary constraint networks, GAC is simply referred as AC (Arc Consistency).

Definition 3. Let $\mathcal{P} = (X, C)$ be a DCN. A pair (x, a) , with $x \in X$ and $a \in \text{dom}(x)$, is generalized arc-consistent (GAC) iff $\forall c \in C \mid x \in \text{scp}(c)$, there exists a support of (x, a) in c , i.e. a tuple $t \in \text{rel}(c)$ such that $t[x] = a$ and $t[y] \in \text{dom}(y) \forall y \in \text{scp}(c)$ ². \mathcal{P} is GAC iff $\forall x \in X, \text{dom}(x) \neq \emptyset$ and $\forall a \in \text{dom}(x), (x, a)$ is GAC.

We will denote by $GAC(\mathcal{P})$ the constraint network obtained after enforcing GAC on \mathcal{P} . Inconsistency proved when applying GAC is denoted by $GAC(\mathcal{P}) = \perp$.

2.4 From Qualitative to Discrete Constraints Networks

In this paper, we propose to use an encoding to map a qualitative constraint network \mathcal{N} into a discrete one \mathcal{P} . Each constraint of \mathcal{N} is mapped to a variable of \mathcal{P} whose domain corresponds to the atomic relations of the constraint (and, as a consequence, a subset of B), and each triple of constraints of \mathcal{N} is mapped to a ternary constraint of \mathcal{P} such that the associated relation contains all valid 3-tuples satisfying the weak composition. More formally, we obtain:

Definition 4. Let $\mathcal{N} = (V, Q)$ be a QCN. $T_{DCN}(\mathcal{N})$ is the DCN $\mathcal{P} = (X, C)$ defined as follows:

- for each pair of variables $v_i, v_j \in V$ with $1 \leq i \leq j \leq n$, X contains a variable x_{ij} such that $\text{dom}(x_{ij}) = Q_{ij}$;
- for each triple of variables $v_i, v_j, v_k \in V$ with $1 \leq i < k < j \leq n$, C contains a ternary constraint c_{ijk} such that $\text{scp}(c_{ijk}) = \{x_{ij}, x_{ik}, x_{kj}\}$ and $\text{rel}(c_{ijk}) = \{(a, b, c) \in B^3 : a \in b \circ c\}$.

The idea of mapping qualitative networks into discrete ones is quite natural, and has been formalized in [26,12]. Interestingly, there are some relationships between the two frameworks. For example, if a QCN \mathcal{N} is consistent, then $T_{DCN}(\mathcal{N})$ is consistent [12]. Unfortunately, the encoding is not complete for some qualitative calculi (e.g. the cyclic interval algebra [18,4]): the qualitative network \mathcal{N} can be

² $t[x]$ denotes the value assigned to x in t .

inconsistent whereas the discrete network $T_{DCN}(\mathcal{N})$ is consistent. Nevertheless, we have the following weaker property: if $T_{DCN}(\mathcal{N})$ is consistent then \mathcal{N} admits a \circ -closed scenario. Besides, introducing the concept of so-called *nice* qualitative calculus, i.e. a calculus for which a scenario is consistent if and only if it is \circ -closed, we can establish that: a QCN \mathcal{N} defined in a *nice* qualitative calculus is consistent iff $T_{DCN}(\mathcal{N})$ is consistent. It is important to remark that many qualitative calculi are nice, and in particular the well-known Allen’s calculus.

It is possible to obtain a qualitative network from a discrete network using the following operator T_{QCN} .

Definition 5. Let $\mathcal{N} = (V, Q)$ be a QCN and $\mathcal{P} = (X, C)$ be a DCN such that $\mathcal{P} \preceq T_{DCN}(\mathcal{N})$. $T_{QCN}(\mathcal{P})$ is the QCN (V, Q') defined by $Q'_{ij} = dom(x_{ij})$ and $Q'_{ji} = (Q'_{ij})^\sim$ for all $1 \leq i \leq j \leq n$.

This operator is useful to show the connections existing between qualitative and discrete local consistencies. Indeed, we can prove [12] that if $T_{DCN}(\mathcal{N})$ is GAC then \mathcal{N} is \circ -closed. As a consequence, a way to obtain the \circ -closure of a QCN is to transform it into a DCN (via T_{DCN}), apply a GAC algorithm and get back the result (via T_{QCN}) under the form of a DCN. This is illustrated in Figure 2.

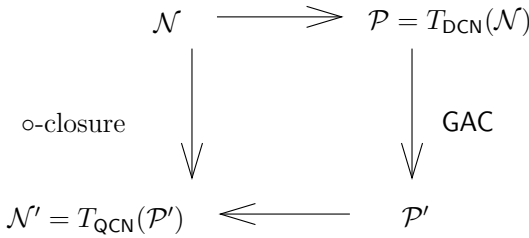


Fig. 2. Relationship between \circ -closure and GAC

The interest of encoding qualitative networks into discrete ones is two-fold. First, we can benefit from some state-of-the-art generic CSP solvers that are freely available. Second, the formalism classically used for qualitative algebra is based on networks whose macro-structure corresponds to complete graphs. Here, introducing relaxation in qualitative networks would require extending current qualitative definitions and properties. The major part of QCN solvers works with matrices as data structures to represent qualitative constraints. For example, QAT (a qualitative algebra toolkit [13]) uses such a data structure. With these solvers, the representation of a relaxed QCN is not a trivial task and implies heavy changes concerning the methods used for reasoning.

3 Relaxing Qualitative Constraints Networks

There is a particular relation in any qualitative algebra: the total relation that we will denote by ψ . This relation, which is such that $\psi = B$, represents the fact

that we have no information about the configuration of any two variables. When solving a QCN, ψ relations may represent an overhead for the resolution since they must be taken into account while not participating (at least, initially) to filter the search space. This is why we are going to propose to relax qualitative constraint networks by simply discarding all ψ relations.

In order to make our approach quite general, we introduce relaxation and restriction with respect to a subset of relations $R \subseteq 2^B$ (even if we will choose $R = \psi$ for our experimentation). From now on, we will consider given the qualitative calculus as well as R (assumed to be closed for the converse operation). Then, we present a general scheme, as well as an algorithm, that can be followed when the qualitative calculus respects some conditions.

3.1 Theoretical Results

To perform our relaxation, we propose a generalization of the mapping operator introduced in Definition 4. The relaxation involves only taking into account variables (of the discrete network) whose domain does not belong to R . More formally, the definition of the new operator, denoted by T_{DCN}^{-R} , is given by:

Definition 6. Let $\mathcal{N} = (V, Q)$ be a QCN. The discrete relaxation $T_{DCN}^{-R}(\mathcal{N})$ of \mathcal{N} is the DCN $\mathcal{P} = (X, C)$ defined as follows:

- for each pair of variables $v_i, v_j \in V$ with $1 \leq i \leq j \leq n$ such as $Q_{ij} \notin R$, X contains a variable x_{ij} such that $dom(x_{ij}) = Q_{ij}$;
- for each triple of variables $v_i, v_j, v_k \in V$ with $1 \leq i < k < j \leq n$, C contains a ternary constraint C_{ijk} , such that $scp(C_{ijk}) = \{x_{ij}, x_{ik}, x_{kj}\}$ and $rel(c_{ijk}) = \{(a, b, c) \in B^3 : a \in b \circ c\}$, iff x_{ij}, x_{ik} and x_{kj} belong to X .

It is immediate to see that T_{DCN}^{-R} is equivalent to T_{DCN} when $R = \emptyset$. Also, $T_{DCN}^{-R}(\mathcal{N})$ is clearly a sub-network of $T_{DCN}(\mathcal{N})$.

The following proposition shows that it is possible to exploit discrete relaxations to prove the unsatisfiability of a qualitative network. The proof is immediate since $T_{DCN}^{-R}(\mathcal{N})$ is a sub-network of $T_{DCN}(\mathcal{N})$ which is equivalent to \mathcal{N} with respect to satisfiability.

Proposition 1. Let \mathcal{N} be a QCN. If $T_{DCN}^{-R}(\mathcal{N})$ is unsatisfiable then \mathcal{N} is unsatisfiable.

Figure 3 shows an illustration of discrete relaxations of a qualitative constraint network (from Allen’s calculus). On the left, we have the discrete network that corresponds to the direct mapping (according to Definition 4) of the QCN presented at the top of the figure: each constraint becomes a variable and each triple of variables becomes a ternary constraint. On the right, we have the discrete network that corresponds to the discrete relaxation based on the total relation. The variable $x_{2,3}$ has been discarded since it corresponds to the total relation $Q_{2,3}$. Consequently, ternary constraints that may involve $x_{2,3}$ have been discarded too. Trivially, both networks are unsatisfiable. Indeed, we have v_1 before v_4 and v_1 after v_4 . Hence, in the relaxed discrete network, $dom(x_{1,4})$ is restricted to

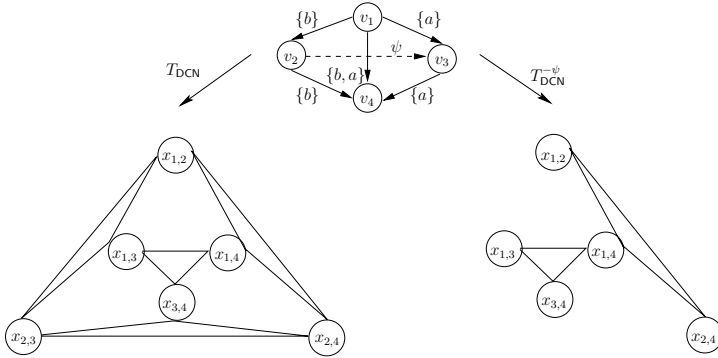


Fig. 3. Illustration of discrete encodings (with and without relaxation)

the *before* value (using the ternary constraint $c_{1,2,4}$) and also to the *after* value (using the ternary constraint $c_{1,3,4}$).

Once a discrete network using T_{DCN}^{-R} has been generated, it is possible to use any CSP solver to find solutions, if any. As mentioned above, if no solution is found, it means that the initial qualitative network is unsatisfiable. However, for any found solution I (i.e. consistent instantiation), it may be interesting to exploit it in the qualitative network. Of course, in the general case, each solution does not correspond to a piece of a consistent scenario of the qualitative network, but by considering each of them in turn, it is possible to render the approach complete. Indeed, if no consistent scenario can be built from all solutions of the discrete relaxation, the qualitative network is proved to be unsatisfiable.

To do this, we need to propose a generalization of the mapping operator introduced in Definition 5.

Definition 7. Let $\mathcal{N} = (V, Q)$ be a QCN, R be a subset of 2^B and $\mathcal{P} = (X, C)$ be a DCN such that $\mathcal{P} \preceq T_{DCN}^{-R}(\mathcal{N})$. The qualitative restriction $T_{QCN}^{+R}(\mathcal{P})$ of \mathcal{P} is the QCN $\mathcal{N}' = (V, Q')$ defined by:

- $\forall 1 \leq i \leq n, Q'_{ii} = \{\text{Id}\};$
- $\forall 1 \leq i < k < j \leq n, Q'_{ij} = \text{dom}(x_{ij})$ and $Q'_{ji} = (Q'_{ij})^\sim$ if $x_{ij} \in X$, and $Q'_{ij} = Q_{ij}$ and $Q'_{ji} = Q_{ji}$, otherwise.

We can apply this operator to any solution found in \mathcal{P} . Indeed, by considering I as a DCN (the domain of each variable being reduced to a single value), we can build the qualitative restriction of I and then obtain a qualitative network \mathcal{N}' . It is interesting to note that in this case, the constraints of \mathcal{N}' are basically composed of relations of R as well as atomic relations. This can contribute to facilitate our task (i.e. determining the satisfiability/unsatisfiability of \mathcal{N}). Indeed, there exists a class $\mathcal{H} \subset 2^B$ of relations, called ORD-Horn relations [25], such that, for some qualitative calculi, weak composition closure is complete with respect to satisfiability. QCN only composed of ORD-Horn relations are called ORD-Horn QCN. We have the following proposition.

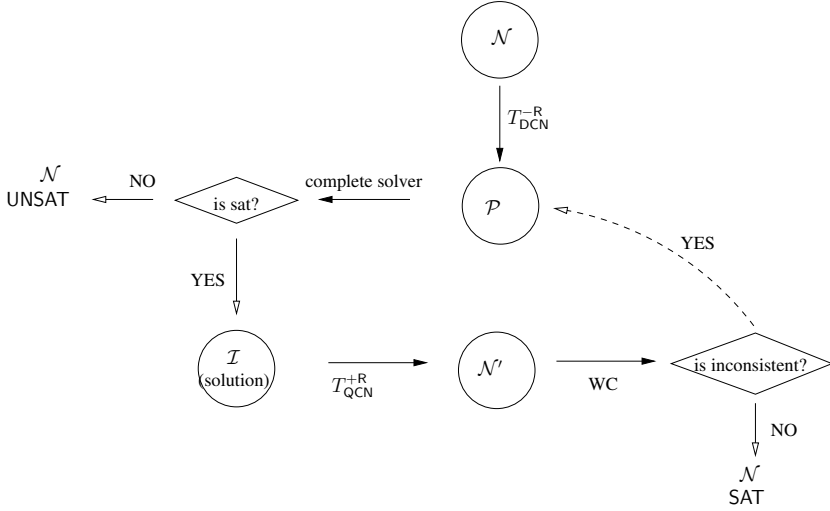


Fig. 4. A general scheme to determine the satisfiability of a QCN \mathcal{N}

Proposition 2. *Let \mathcal{N} be a QCN and I be a solution of $T_{DCN}^{-R}(\mathcal{N})$. If R is a subset of ORD-Horn relations and $T_{QCN}^{+R}(I)$ is closed by weak composition, then \mathcal{N} is satisfiable.*

The proof can be derived from the elements introduced above. This proposition can be directly exploited by some qualitative calculi, e.g. the Allen's one. Indeed, for the Allen's calculus, the set of ORD-Horn relations is closed with respect to composition, intersection and converse. It implies that, applying a weak composition closure algorithm on an ORD-Horn network yields another ORD-Horn network. In other words, for the Allen's algebra, Propositions 1 and 2 provide us with an original way to determine the satisfiability of a qualitative network by using two simple mapping operators, a CSP solver and a weak composition closure algorithm.

3.2 General Scheme

A general scheme about the exploitation of discrete networks to deal with the relaxation of qualitative networks is given in Figure 4. This scheme holds if the qualitative calculus is such that all atomic relations are ORD-Horn relations and the weak composition closure is complete with respect to satisfiability. Remark that the closure of the set of ORD-Horn relations for composition and intersection may improve the applicability of this scheme.

The aim of the scheme is to determine the satisfiability of a given qualitative network \mathcal{N} . Using the operator T_{DCN}^{-R} (see Definition 6), we first obtain a discrete network \mathcal{P} . Then, we use a complete solver to search for a solution. If no solution is found, then \mathcal{N} is proved to be unsatisfiable (see Proposition 1). Otherwise, a solution I is returned by the solver. Using the operator T_{QCN}^{+R} (see Definition

7), we can then obtain a new qualitative network \mathcal{N}' . If this network can be enforced to be \circ -closed, it means that the satisfiability of \mathcal{N} has been proved (see Proposition 2). If this is not the case, we just ask the complete solver to search for the next solution.

Algorithm 1. checkSatisfiability

```

Data: QCN  $\mathcal{N}$ 
Result: the satisfiability of  $\mathcal{N}$ 
begin
   $\mathcal{P} = T_{\text{DCN}}^{-\psi}(\mathcal{N})$ 
  /* we assume that solve( $\mathcal{P}$ ), called iteratively, returns the
     solutions of  $\mathcal{P}$ , one by one */
  solution  $\leftarrow$  solve( $\mathcal{P}$ )
  while solution  $\neq$  null do
    if GAC(solution  $\cap$   $T_{\text{DCN}}(\mathcal{N})$ )  $\neq \perp$  then
      L return SAT
    solution  $\leftarrow$  solve( $\mathcal{P}$ )
  return UNSAT
end

```

We will instantiate this general scheme by using the total relation as relaxation (i.e. by choosing $R = \psi$) and considering the Allen’s calculus whose atomic and total relations belong to the set of ORD-Horn relations, which consequently satisfies the previous requirements. More precisely, we will use the function *checkSatisfiability* depicted by Algorithm 1. The main difference between this algorithm and the scheme presented above is the fact that, instead of using the operator $T_{\text{QCN}}^{+\text{R}}$ and the \circ -closure operation, we use a GAC algorithm. Indeed, if we consider the network obtained by restricting $T_{\text{DCN}}(\mathcal{N})$ with respect to the last found solution, and if we apply GAC, then we obtain the same result as the one obtained by applying the \circ -closure on $T_{\text{QCN}}^{+\text{R}}(\text{solution})$ (see Figure 2).

4 Experiments

To study the practical interest of our proposed relaxation framework, some preliminary experiments on unsatisfiability detection have been conducted using total relations relaxation. Some qualitative constraint networks have been randomly generated and converted to discrete constraint networks using the qualitative algebra toolkit QAT [13]. We have limited our attention to random networks because of the absence of structured qualitative instances. Then, Abscon, a state-of-the-art generic CSP solver (see <http://www.cril.univ-artois.fr/CPAI06>) has been run on the obtained discrete instances. Note that the satisfiability detection part of our general framework is currently under development.

In our experiments, as path consistency (which corresponds to weak composition) on qualitative constraint networks can lead to the elimination of some total relations (replaced by implied ones), two different relaxation modes have

been considered. For the first one, relaxation corresponds to the elimination of all total relations: this will be called strong relaxation. For the second one, relaxation (elimination of total relations) is done after achieving weak composition: this will be called weak relaxation. Of course, the strong relaxation can eliminate more total relations than the weak one. Using these two relaxation modes, an experimental comparison has been conducted against a direct resolution (no relaxation).

To generate random QCN instances, we have used three parameters : the number of variables (N), the density of non-total relations (D) and the average number of basic relations for those constraints (L). The generated QCN instances are composed of 50 variables built from the well-known Allen's calculus (which is composed of 13 basic relations). For each instance, the number of atomic relations in each constraint has been fixed to $L_1 = 3.25$ and $L_2 = 6.5$ while the density of non-total relations has been varied from 0.01 to 0.99 with a step of 0.01. For each parameter setting, 100 networks have been generated.

Comparison is done according to the percentage of detected unsatisfiable instances and average cpu time (in ms) needed to solve such instances. The first measure is used to show the precision of our relaxation framework i.e. the highest this percentage is, the better our approximation is.

Figure 4 indicates that for L_1 (figure on top) and L_2 (figure on bottom), the strong relaxation is far less efficient, in term of precision (i.e. detected unsatisfiable instances), than the weak one. Note that for L_1 , 100% of unsatisfiable instances have been detected by the weak relaxation, and that for L_2 , the gap between weak and strong relaxations is increasing. It then appears that applying weak composition before relaxation improves the precision of our unsatisfiability detection method. Note that these experiments are only presented around the threshold which is the most interesting area. Of course, beyond the highest density mentioned in Figure 4, the accuracy of both relaxation methods always reaches 100%.

It is interesting to see that in term of cpu time, the two relaxation modes present totally different behaviours. This can be observed for both strong and weak relaxations in Figures 6 and 7, respectively. One could imagine that strong relaxation is faster than weak relaxation on detected unsatisfiable instances, but here, we have to be aware that the average computation is not done on the same basis. Indeed, we make our computation by only keeping the instances that were detected unsatisfiable by, on the one hand (Figure 6), both the strong relaxation and the direct resolution, and on the other hand (Figure 7), both the weak relaxation and the direct resolution. Note that, as the density increases, the two relaxations tend to be very close to the behaviour of the direct resolution. Also, remark that in Figure 7 (on bottom), at a density set to 0.18, we obtain a significant gain in term of cpu time when using the weak relaxation.

Finally, to make our experimental protocol more significant, we have conducted another series of experiments on a sample of hard QCN instances above the threshold. To this end, a number of atomic relations per constraint ($L_3 = 9.75$) and a non-total relations density ($D = 0.65$) have been chosen such that the generated networks are difficult to solve. As we can observe in

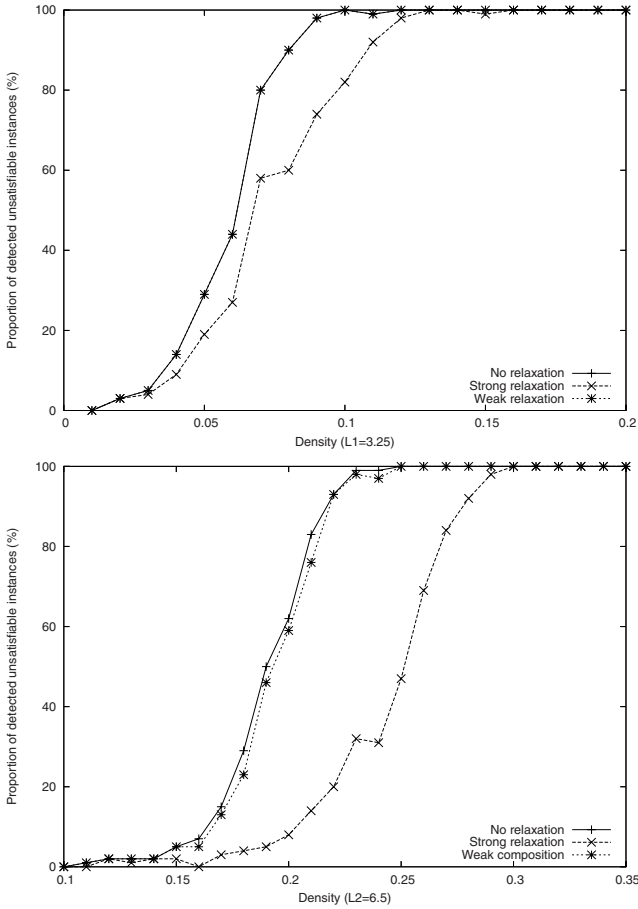


Fig. 5. Proportion of instances detected as unsatisfiable

Table 1. Results on a sample of difficult instances

Method	No relaxation	Strong relaxation	Weak relaxation
# instances	65	65	65
# Time Out (TO)	32	16	16
# Sat	0	0	0
# Unsat	33	49	49
# Time Out / # instances	49,23%	24,62%	24,62%
Total Time (33 unsat) in s	14,572	3,765	4,016
Avg Time (33 unsat) in s	441.5	114.1	121.7
Ratio	0,00%	74,16%	72,43%

Table 1, our approach is very efficient for detecting the unsatisfiability of these hard instances. Indeed, without relaxation, 49.23% of instances are not solved within the allowed cpu time (1200s), whereas on the same networks only 24.62%

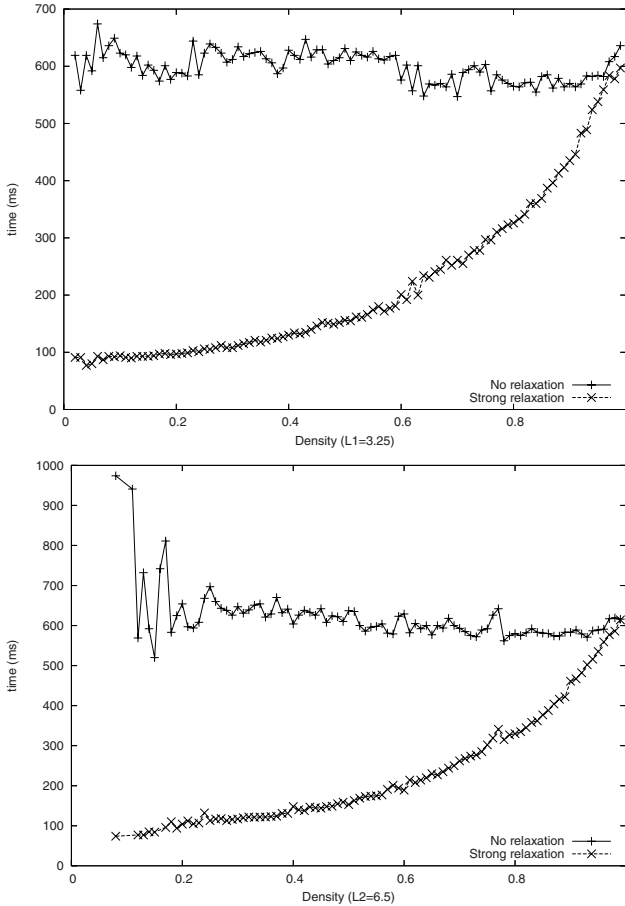


Fig. 6. Average cpu time to detect unsatisfiability (strong relaxation)

of instances are not detected to be unsatisfiable by our approach. Interestingly enough, on such detected instances a huge gain (up to a four-fold improvement) is obtained with respect to cpu time. On these hard instances, the weak and strong relaxations present very close performances.

5 Future Works and Conclusions

In this paper, a general and complete relaxation framework for qualitative reasoning is proposed. It allows the user to consider any kind of relaxation and any type of relations. Its originality comes from the fact that we exploit a mapping towards discrete constraint networks. Considering total relations for relaxation, promising preliminary results have been obtained with respect to unsatisfiability.

The generality and flexibility behind our approach offer some rooms for further improvements. It can be strengthened by integrating results from constraint

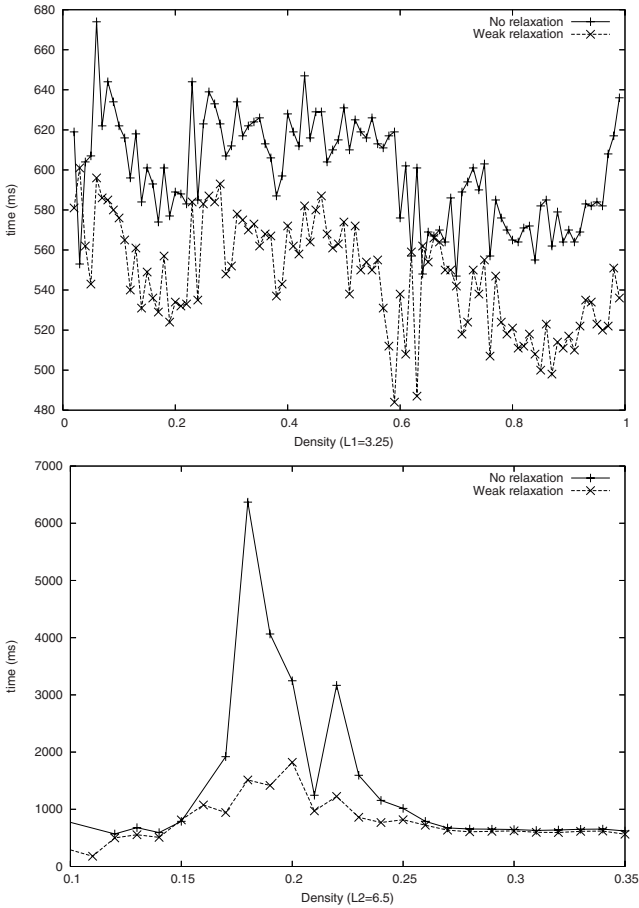


Fig. 7. Average cpu time to detect unsatisfiability (strong relaxation)

satisfaction and satisfiability problems. For example, the completeness can be efficiently addressed by exploiting advanced methods in satisfiability problems (e.g. randomisation and restarts [17]). We can also imagine the integration of different forms of restriction in order to reintroduce some of the previous relaxed relations. This could be done, for example, by using constraint weighting [7] to select the best relations that might be added.

References

1. Allen, J.F.: An interval-based representation of temporal knowledge. In: Proceedings of IJCAI’81, pp. 221–226 (1981)
2. Allen, J.F.: Maintaining Knowledge about Temporal Intervals. Communications of the ACM 26(11), 832–843 (1983)

3. Balbiani, P., Condotta, J.F., Fariñas del Cerro, L.: A model for reasoning about bidimensional temporal relations. In: Proceedings of KR'98, pp. 124–130 (1998)
4. Balbiani, P., Osmani, A.: A model for reasoning about topologic relations between cyclic intervals. In: Proceedings of KR'00, pp. 378–385 (2000)
5. Bistarelli, S., Codognet, P., Rossi, F.: An abstraction framework for soft constraints and its relationship with constraint propagation. In: Choueiry, B.Y., Walsh, T. (eds.) SARA 2000. LNCS (LNAI), vol. 1864, pp. 71–86. Springer, Heidelberg (2000)
6. Bistarelli, S., Codognet, P., Rossi, F.: Abstracting soft constraints: framework, properties, examples. *Artificial Intelligence* 139, 175–211 (2002)
7. Boussemart, F., Hemery, F., Lecoutre, C., Sais, L.: Boosting systematic search by weighting constraints. In: Proceedings of ECAI'04, pp. 146–150 (2004)
8. Cadoli, M.: Tractable Reasoning in Artificial Intelligence. LNCS, vol. 941, Springer, Heidelberg (1995)
9. Caseau, Y.: Abstract interpretation of constraints on order-sorted domains. In: Proceedings of ISLP'91, pp. 435–452 (1991)
10. Choueiry, B., Faltings, B., Weigel, R.: Abstraction by interchangeability in resource allocation. In: Proceedings of IJCAI'95, pp. 1694–1710 (1995)
11. Choueiry, B., Noubir, G.: On the computation of local interchangeability in discrete constraint satisfaction problems. In: Proceedings of AAAI'98, pp. 326–333 (1998)
12. Condotta, J.F., Dalmeida, D., Lecoutre, C., Sais, L.: From qualitative to discrete constraint networks. In: Freksa, C., Kohlhase, M., Schill, K. (eds.) KI 2006. LNCS (LNAI), vol. 4314, pp. 54–64. Springer, Heidelberg (2007)
13. Condotta, J.F., Ligozat, G., Saade, M.: The QAT: A Qualitative Algebra Toolkit. In: Proceedings of TIME'2006, pp. 69–77 (2006)
14. Cousot, P., Cousot, R.: Abstract interpretation frameworks. *Logic and Computation* 2(4), 447–511 (1992)
15. Freuder, E., Sabin, D.: Interchangeability supports abstraction and reformulation for constraint satisfaction. In: Proceedings of SARA'95 (1995)
16. Giunchiglia, F., Walsh, T.: A theory of abstraction. *Artificial Intelligence* 56(2-3), 323–390 (1992)
17. Gomes, C.P., Selman, B., Crato, N., Kautz, H.: Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning* 24, 67–100 (2000)
18. Hornsby, K., Egenhofer, M.J., Hayes, P.J.: Modeling cyclic change. In: Proceedings of REIS'99, pp. 98–109 (2006)
19. Isli, A., Cohn, A.G.: A new approach to cyclic ordering of 2D orientations using ternary relation algebras. *Artificial Intelligence* 122(1–2), 137–187 (2000)
20. Jussien, N.: Relaxation de Contraintes pour les problèmes dynamiques. PhD thesis, Université de Rennes I (1997)
21. Jussien, N., Boizumault, P.: Implementing constraint relaxation over finite domains using ATMS. In: Jampel, M., Maher, M.J., Freuder, E.C. (eds.) Over-Constrained Systems. LNCS, vol. 1106, Springer, Heidelberg (1996)
22. Ligozat, G.: Reasoning about cardinal directions. *Journal of Visual Languages and Computing* 1(9), 23–44 (1998)
23. Marriott, K.: Frameworks for abstract interpretation. *Acta Informatica* 30, 103–129 (1993)
24. Merchez, S., Lecoutre, C., Boussemart, F.: Abstraction de réseaux de contraintes. *Revue d'Intelligence Artificielle* 20(1), 31–62 (2006)
25. Nebel, B., Bürckert, H.J.: Reasoning About Temporal Relations: A Maximal Tractable Subclass of Allen's Interval Algebra. *Journal of the ACM* 42(1), 43–66 (1995)

26. Pham, D.N., Thornton, J., Sattar, A.: Modelling and solving temporal reasoning as propositional satisfiability. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 117–131. Springer, Heidelberg (2005)
27. Pujari, A.K., Kumari, G.V., Sattar, A.: Indu: An interval and duration network. In: Proceedings of the Australian Joint Conference on Artificial Intelligence, pp. 291–303 (1999)
28. Randell, D.A., Cui, Z., Cohn, A.G.: A spatial logic based on regions and connection. In: Proceedings of KR'92, pp. 165–176 (1992)
29. Selman, B., Kautz, H.: Knowledge compilation and theory approximation. *Journal of the ACM* 43(2), 193–224 (1996)
30. Shrag, R., Miranker, D.: Abstraction and the csp phase transition boundary. In: Proceedings of the 4th International Symposium on Artificial Intelligence and Mathematics, pp.138–141, 1996.
31. Yokoo, M.: Constraint relaxation in distributed constraint satisfaction problems. In: Proceedings of ICTAI'03, pp. 56–63 (1993)

Dynamic Domain Abstraction Through Meta-diagnosis

Johan de Kleer

Palo Alto Research Center,
3333 Coyote Hill Road, Palo Alto, CA 94304 USA
dekleer@parc.com

Abstract. One of the most powerful tools designers have at their disposal is abstraction. By abstracting from the detailed properties of a system, the complexity of the overall design task becomes manageable. Unfortunately, faults in a system need not obey the neat abstraction levels of the designer. This paper presents an approach for identifying the abstraction level which is as simple as possible yet sufficient to address the task at hand. The approach chooses the desired abstraction level through applying model-based diagnosis at the meta-level, i.e., to the abstraction assumptions themselves.

Keywords: Abstraction, diagnosis, qualitative reasoning, model-based reasoning.

1 Introduction

Of the many tools designers have at their disposal, abstraction is one of the most powerful. By abstracting from the detailed properties of a system, the complexity of the overall design task becomes manageable. For example, a computer engineer can focus on the logic level without concern for the properties of the individual transistors which make up a particular gate, and a chip designer can layout a chip without being concerned with the fabrication steps needed to construct it. Abstraction allow designers to partition concerns into independent black boxes and is one of the most important ideas underlying the design of modern technology.

Unfortunately, faults in a system need not obey the neat abstraction levels of the designer. A fault in a few transistors can cause an Intel Pentium processor to generate an occasional incorrect floating point result. To understand this fault requires transcending the many abstraction levels between software and hardware. A PC designer can focus on functional layout without being concerned about the physical layout and its thermal properties. However, a technician must determine that the processor crashed because dust sucked into the processor fan clogged the heatsink and allowed the processor temperature to rise to such a dangerous level that the PC automatically shut down. As a consequence diagnostic reasoning is inherently messy and complex, as it involves crossing abstraction boundaries never contemplated by the designers.

Existing model-based reasoning has addressed a number of types of abstraction.

- Range abstraction. The ranges of variables are abstracted, e.g., instead of a continuous quantity it might be represented by the qualitative values of $-$, 0 or $+$. [12,3,4]
- Structural abstraction. Groups of components are abstracted to form hierarchies [5,6].
- Model selection. Approaches to choosing among a collection of hand-constructed models [7,8]

In this paper we present a new type of abstraction (*domain* abstraction): changing the physical principles which underlie models, such as moving from the 0/1 level to currents and voltages and providing a systematic approach to choosing the appropriate domain for the diagnostic task.

Choosing the right domain abstraction level requires balancing two opposing desiderata. Reasoning at the highest abstraction level is the simplest. Unfortunately, it may be inadequate to analyze or troubleshoot the system. Instead, the system needs to be analyzed at a more detailed level. On the other hand, reasoning at too low of a level can require enormously more computational resources and difficult to obtain parameters, and it generates more complicated analyses. As Albert Einstein reportedly said: “Make everything as simple as possible, but not simpler.”

Technicians expect that systems are non-intermittent and that the schematic is an accurate description of the physical system. Consider the simple analog circuit of Fig. 1. Suppose a technician measures the current to be 0 ampere (1 is expected), which leads to an inference that the resistor is faulty, but repeating the measurement gives 1 ampere. Either the resistor is intermittent or there is a fault in the connections. The technician must change abstraction level to diagnose this system further by, for example, checking the connections or further tests on resistor R itself.

Consider a circuit of three logic inverters in sequence, with its output feedback to its input (Fig. 2). At the usual gate level of analysis, an inverter simply complements its input. This circuit has no inputs, so we need to consider the possible values at the connecting nodes. Assume the input to inverter A is 0. Its

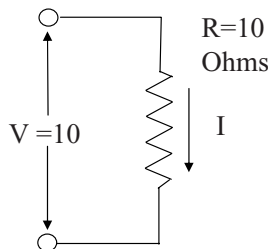


Fig. 1. Simple analog diagnosis problem

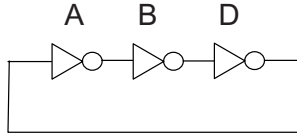


Fig. 2. A very simple circuit which yields a contradiction when analyzed as combinational logic; yet its a perfectly reasonable fault-free circuit with well-defined behavior

output must be 1. The output of B must be 0. The output of D must be 1. This is impossible, as we assumed it was 0. Conversely, assume the input to inverter A is 1. Its output must be 0. The output of B must be 1. The output of D must be 0. This is impossible, as we assumed it was 1. Therefore, the input of inverter A can neither be 0 or 1. Also, the inputs of inverters B or D cannot be 0 or 1. Somehow the circuit is contradictory when modeled as logic gates. Thus, one of the components A , B or D must be faulted. Suppose the technician chooses to systematically remove and check each of these three components for proper functioning. If each component is confirmed to be correct, the technician has encountered an impasse.

Analyses that result in contradictions are the most important indicator that the level of abstraction used is too simplistic. In this paper we present a general reasoner which automatically descends abstraction layers to perform needed analyses, and which does not descend abstraction levels needlessly. This approach is broadly applicable, but we explore these ideas in the context of digital circuits with messier models that include failures in connections, intermittents, and temporal behaviors.

2 Meta-diagnosis

Fig. 3 characterizes the basic architecture of a typical model-based, component-based diagnosis engine. Given the component topology (e.g., the schematic for analog circuits), component models (e.g., resistors obey ohm's law) and observations (e.g., the voltage across resistor $R6$ is 4 volts), it computes diagnoses which explain all the observations. Observations inconsistent with expectations guide the discovery of diagnoses. When the MBD engine can find no diagnoses it signals a failure.

Suppose we need to troubleshoot the circuit of Fig. 2. Most diagnosis systems would immediately conclude that some subset of the components $\{A, B, D\}$ is faulted. However, testing each inverter individually demonstrates that all the components are good. As a consequence, most algorithms would report an unresolvable contradiction.

The architecture of Fig. 4 includes two model-based diagnosis engines. The top model is used to identify the best abstraction level, and the bottom model performs the actual system diagnosis. This composite architecture has the same inputs as the basic architecture with one additional input: the abstraction library.

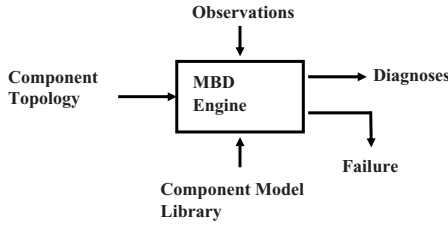


Fig. 3. Basic architecture of a model-based diagnosis engine

The ‘Applicable Models’ module identifies all the applicable abstractions for the component topology. The ‘Modeler’ module uses the preferred meta-diagnosis to construct conventional model-based diagnosis models using the ‘Component Model Library.’

Consider the example of Fig. 2. The component topology is simply the circuit schematic as before. The system observations are as before (e.g., the output of *A* is 1). The component model library contains different models for gate behavior (e.g., boolean, analog, thermal, temporal, etc.). The new input, the abstraction library, is the set of all possible abstractions. Instead of a usual component topology, the abstraction MBD engine is provided with a set of possible abstractions applicable to the given system to be diagnosed. Initially, there are no meta-observations, so the preferred diagnosis is the one at the most abstract level (analogous to all components working). Therefore, the domain MBD engine

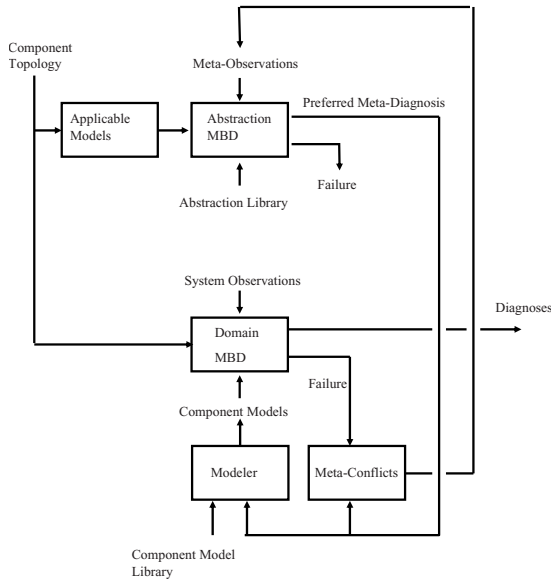


Fig. 4. Architecture of an abstraction-based, model-based diagnosis engine

will perform diagnosis in the usual way with the most abstract models. Suppose each gate is physically checked, leading to the observations that A , B and D are working correctly. The domain model-based engine now fails as it has found an unresolvable contradiction. This invokes the abstraction MBD engine as an observation. As analysis proceeds, the preferred meta-diagnosis will descend abstraction levels. For the purposes of this paper, the preferred meta-diagnosis is one minimal cardinality meta-diagnosis.

3 Formalization

This section summarizes the formal framework for model-based diagnosis we use in the rest of the paper [9,10]. In order to distinguish between domain and abstraction AB literals, we state the usual framework in terms of domain AB_d literals.

Definition 1. *A system is a triple $(SD, COMPS, OBS)$ where:*

1. SD , the system description, is a set of first-order sentences.
2. $COMPS$, the system components, is a finite set of constants.
3. OBS , a set of observations, is a set of first-order sentences.

In Fig. 3 SD is the component topology and component model library, and $COMPS$ is the set of components in the component topology.

Definition 2. *Given two sets of components C_p and C_n define $\mathcal{D}_d(C_p, C_n)$ to be the conjunction:*

$$\left[\bigwedge_{c \in C_p} AB_d(c) \right] \wedge \left[\bigwedge_{c \in C_n} \neg AB_d(c) \right].$$

Where $AB_d(x)$ represents that the component x is AB_{normal} (faulted).

A diagnosis is a sentence describing one possible state of the system, where this state is an assignment of the status normal or abnormal to each system component.

Definition 3. *Let $\Delta \subseteq COMPS$. A diagnosis for $(SD, COMPS, OBS)$ is $\mathcal{D}_d(\Delta, COMPS - \Delta)$ such that the following is satisfiable:*

$$SD \cup OBS \cup \{\mathcal{D}_d(\Delta, COMPS - \Delta)\}$$

Definition 4. *An AB_d -literal is $AB_d(c)$ or $\neg AB_d(c)$ for some $c \in COMPS$.*

Definition 5. *An AB_d -clause is a disjunction of AB_d -literals containing no complementary pair of AB_d -literals.*

Definition 6. *A conflict of $(SD, COMPS, OBS)$ is an AB_d -clause entailed by $SD \cup OBS$.*

3.1 Formalizing Abstraction

The abstraction MBD is defined analogously:

Definition 7. *An abstraction system is a triple (SD, ABS, OBS) where:*

1. *SD, constraints among possible abstractions, is a set of first-order sentences.*
2. *ABS, the applicable abstractions, is a finite set of constants.*
3. *OBS, a set of meta-observations, is a set of first-order sentences.*

Definition 8. *Given two sets of abstractions Cp and Cn define $\mathcal{D}_a(Cp, Cn)$ to be the conjunction:*

$$\left[\bigwedge_{c \in Cp} AB_a(c) \right] \wedge \left[\bigwedge_{c \in Cn} \neg AB_a(c) \right].$$

Where $AB_a(x)$ represents that the abstraction x is ABnormal (cannot be used).

A meta-diagnosis is a sentence describing one possible state of the system, where this state is an assignment of the status normal or abnormal to each system component.

Definition 9. *Let $\Delta \subseteq ABS$. A meta-diagnosis for (SD, ABS, OBS) is $\mathcal{D}_a(\Delta, ABS - \Delta)$ such that the following is satisfiable:*

$$SD \cup OBS \cup \{\mathcal{D}_a(\Delta, ABS - \Delta)\}$$

Definition 10. *An AB_a -literal is $AB_a(c)$ or $\neg AB_a(c)$ for some $c \in ABS$.*

Definition 11. *An AB_a -clause is a disjunction of AB_a -literals containing no complementary pair of AB_a -literals.*

Definition 12. *A meta-conflict of (SD, ABS, OBS) is an AB_a -clause entailed by $SD \cup OBS$.*

4 Example of a Lattice of Models

To illustrate these ideas we use 3 axes of abstraction:

- Model of connections as in [11] which is an improvement over [12][13].
- Model of non-intermittency [14] or intermittency [15]
- Model of time [16].

The corresponding AB_a literals are:

- $\neg AB_a(C)$ represents the abstraction that connections need not be modeled.
- $\neg AB_a(I)$ represents the abstraction that the system is non-intermittent.
- $\neg AB_a(T)$ represents the abstraction that temporal behavior need not be modeled.

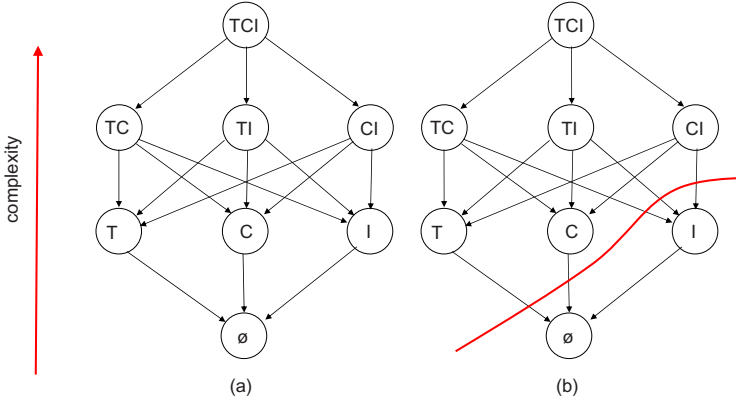


Fig. 5. (a) Meta-Diagnosis lattices for digital gates. T indicates temporal models; C indicates connection models; I indicates intermittent models. (b) The meta-conflict $AB_a(T) \vee AB_a(C)$ rules out all meta-diagnoses below the curved line. The minimal meta-diagnoses are T and C .

Fig. 5(a) shows a portion of the abstraction space for digital circuits along three of the axes of abstraction. This lattice is identical in structure to the ones used in conventional model-based diagnosis for system diagnoses. In conventional model-based diagnosis, each node represents a candidate diagnosis which explains the observations. Each node in Fig. 5(b) represents a candidate meta-diagnosis. The bottom node in the figure represents the meta-diagnosis in which connections, time, and intermittency are not relevant: $\neg AB_a(T) \wedge \neg AB_a(C) \wedge \neg AB_a(I)$. Under the principle that we want to find the simplest meta-diagnosis which explains the observations (and no simpler), we are primarily interested in the minimal diagnoses. For brevity sake, we name meta-diagnoses with the letters corresponding to the abstractions which are AB_a . For example, the meta-diagnosis $\neg AB_a(T) \wedge \neg AB_a(C) \wedge AB_a(I)$ is named by I .

For the example in Fig. 2, analysis immediately detects a contradiction and the meta-conflict: $AB_a(T) \vee AB_a(C)$. (This contradiction cannot depend on $AB_a(I)$ as there is only one observation time so far). Fig. 5(b) illustrates the resulting meta-diagnosis lattice. Every meta-diagnosis below the curve is eliminated. The minimal meta-diagnoses are T and C .

5 Modeling Components

The conventional MBD model for an inverter is (presuming the usual background axioms define the appropriate functions, domains, and ranges):

$$\text{INVERTER}(x) \rightarrow \left[\neg AB_d(x) \rightarrow [in(x, t) = 0 \equiv out(x, t) = 1] \right].$$

As this model presumes connections and temporal behavior need not be modeled, in our new framework it is written as:

$$\neg AB_a(T) \wedge \neg AB_a(C) \rightarrow \left[\text{INVERTER}(x) \rightarrow \left[\neg AB_d(x) \rightarrow [in(x, t) = 0 \equiv out(x, t) = 1] \right] \right].$$

When modeling an inverter as having a delay Δ , the model changes to (labeled T in Fig. 5):

$$AB_a(T) \wedge \neg AB_a(C) \rightarrow \left[\text{INVERTER}(x) \rightarrow \left[\neg AB_d(x) \rightarrow [in(x, t) = 0 \equiv out(x, t + \Delta) = 1] \right] \right].$$

5.1 Connection Models

To model the inverter to accommodate faults in connections, including bridge faults, requires the introduction of new formalisms. What follows is a brief summary of the formalism presented in [11]. Each terminal of a component is modeled with two variables, one which models how the component is attempting to influence its output (roughly analogous to current), and the other which characterizes the result (roughly analogous to voltage). There are 5, mutually inconsistent, qualitative values for the influence of a component on a node (we refer to these as “drivers”).

- $d(-\infty)$ indicates a direct short to ground.
- $d(0)$ pull towards ground (i.e., 0).
- $d(R)$ presents a high (i.e., draws little current) passive resistive load.
- $d(1)$ pull towards power (i.e., 1).
- $d(+\infty)$ indicates a direct short to power.

There are three possible qualitative values for the resulting signal:

- $s(0)$ the result is close enough to ground to be sensed as a digital 0.
- $s(x)$ the result is neither a 0 or 1.
- $s(1)$ the result is close enough to power to be sensed as a digital 1.

Using this formalism produces considerably more detailed component models. We need to expand the $A \equiv B$ in the inverter model to $(A \rightarrow B) \wedge (B \rightarrow A)$. The left half of the inverter model is:

$$\begin{aligned} \neg AB_a(T) \wedge AB_a(C) \rightarrow & \left[\text{INVERTER}(x) \rightarrow \left[\neg AB_d(x) \rightarrow \right. \right. \\ & \left[[s(in(x, t)) = s(0) \rightarrow d(out(x, t)) = d(1)] \right. \\ & \left. \wedge [s(in(x, t)) = s(1) \rightarrow d(out(x, t)) = d(0)] \right. \\ & \left. \left. \wedge d(in(x, t)) = d(R) \wedge [d(out(x, t)) = d(0) \vee d(out(x, t)) = d(1)] \right] \right]. \end{aligned}$$

We need explicit models to describe how the digital signal at a the node is determined from its drivers. Let $R(v)$ be the resulting signal at node v and $S(v)$ be the collection of drivers of node v . Intuitively, the model for a node is:

- If $d(-\infty) \in S(v)$, then $R(v) = s(0)$.
- If $d(+\infty) \in S(v)$, then $R(v) = s(1)$.
- If $d(0) \in S(v)$, then $R(v) = s(0)$.
- Else, if all drivers are known, and the preceding 3 rules do not apply, then $R(v) = s(1)$.

The resulting model for the node x will depend on $\neg AB_d(x)$ and $AB_a(C)$.

Modeling the inverter to more accurately describe both temporal and causal behavior (labeled TC in Fig. 5):

$$\begin{aligned}
 AB_a(T) \wedge AB_a(C) \rightarrow & \left[\text{INVERTER}(x) \rightarrow \left[\neg AB_d(x) \rightarrow \right. \right. \\
 & \left. \left[[s(\text{in}(x, t)) = s(0) \rightarrow d(\text{out}(x, t + \Delta)) = d(1)] \right. \right. \\
 & \left. \wedge [s(\text{in}(x, t)) = s(1) \rightarrow d(\text{out}(x, t + \Delta)) = d(0)] \right. \\
 & \left. \left. \left. \wedge d(\text{in}(x, t)) = d(R) \wedge [d(\text{out}(x, t + \Delta)) = d(0) \vee d(\text{out}(x, t + \Delta)) = d(1)] \right] \right] \right].
 \end{aligned}$$

The connection models also allow arbitrary bridge faults among circuit nodes. These are described in much more detail in [11].

5.2 Modeling Non-intermittency

Fig. 6 shows an example where assuming non-intermittency improves diagnostic discrimination. The circuit's inputs and outputs are marked with values observed at times: T_1 and T_2 . Note that at T_1 , the circuit outputs a correct value and that at T_2 , the circuit outputs an incorrect one. By assuming the Or gate behaves non-intermittently, we can establish that the Xor gate is faulty as follows:

If Xor is good, then $\text{In}_1(\text{Xor}, T_1) = 1$. This follows from $\text{In}_2(\text{Xor}, T_1) = 0$, $\text{Out}(\text{Xor}, T_1) = 1$ and the behavior of Xor. Similarly, if Xor is good, then $\text{In}_1(\text{Xor}) = 0$ at T_2 . However, if Or behaves non-intermittently, then $\text{In}_1(\text{Xor}, T_2) = 1$. This follows because Or has the same inputs at both T_1 and T_2 and must produce the same output. Thus we have two contradictory predictions for the value of $\text{In}_1(\text{Xor}, T_2)$. Either Xor is faulty or Or is behaving intermittently. Assuming non-intermittency means Xor is faulty.

All the inferences follow from the defining of non-intermittency:

Definition 13. [14] *A component behaves non-intermittently if its outputs are a function of its inputs.*

This definition sanctions the following inference: if an input vector \overline{X} is applied to an intermittent component at time T , and output Z is observed, then in any other observation T' , if \overline{X} is supplied as input, Z will be observed as output.

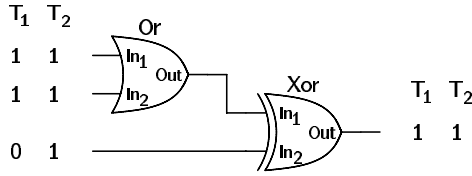


Fig. 6. The power of non-intermittency

For the Or-Xor example, the axioms added are:

$$\forall t. Out(Or, t) = F(Or, In_1(Or, t), In_2(Or, t))$$

$$\forall t. Out(Xor, t) = F(Xor, In_1(Xor, t), In_2(Xor, t))$$

F is a single fixed function for all non-intermittency axioms.

These axioms are implemented in the ATMS/HTMS-based reasoner by deriving prime implicates as follows. At time T_1 :

$$AB_d(Xor) \vee [F(Or, 1, 1) = 1].$$

At time T_2 :

$$AB_d(Xor) \vee [F(Or, 1, 1) = 0].$$

Which combine to yield $AB_d(Xor)$.

In the intermittent case, the observation at T_1 equally weights Xor and Or as being correct. If there were other components in the system not affected by the measurement, the observation at T_1 lowers the posterior fault probabilities of Xor and Or.

5.3 Automatic Generation of Models

The more detailed component models can usually be generated automatically from the most abstract models in a systematic way. In our implementation, the T , C , and I models are automatically derived from the basic \emptyset models by a set of modeling schemas. Consider the most abstract model of an inverter:

$$INVERTER(x) \rightarrow [\neg AB_d(x) \rightarrow [in(x, t) = 0 \equiv out(x, t) = 1]].$$

We use the convention that the function in refers to inputs, and the function out refers to outputs. A non-temporal model can be converted to a simple gate-delay model by replacing every occurrence of $out(x, t)$ (or $out_j(x, t)$) with $out(x, t + \Delta)$.

A non-connection model can be converted to a connection one by first expanding implications, replacing all $in(x, t) = y$ with $s(in(x, t)) = s(y)$ and $d(in(x, t)) = d(R)$ and replacing all $out(x, t) = y$ with $d(out(x, t)) = d(y)$, and adding the usual domain axioms for new variable values.

Non-intermittency requires no change to the component models themselves, but the axioms of Section 5.2 need to be added to the models supplied to the domain MBD.

6 The Meta-diagnosis Loop

6.1 $\emptyset \rightarrow T$

Consider the three inverter example of Fig. 2. The most abstract meta-diagnosis is:

$$\neg AB_a(T) \wedge \neg AB_a(C) \wedge \neg AB_a(I).$$

This meta-diagnosis is supplied to the ‘Modeler’ module for Fig. 4 which chooses the component models at the meta-diagnosis level. The models for all three inverters are given by the TC model of Section 5. This produces a failure because all components are known to be good. The ‘Meta-Conflicts’ module of Fig. 4 will construct the meta-conflict:

$$AB_a(T) \vee AB_a(C).$$

$AB_a(I)$ is trivially excluded from the meta-conflict because non-intermittency inferences can only arise when the system has been observed at multiple times.

The abstraction MBD identifies two minimal meta-diagnoses T and C . If both are equally likely, it arbitrarily picks one. Suppose C is chosen. The C models do not resolve the inconsistency either. Fig. 7 illustrates the following sequence of inferences with the connection models: (1) Assume the input of A is 1, (2) the causal inverter model drives its output down towards 0, (3) the input of gate B presents a high resistance (low-current) load to its node, (4) the connection model sets the node to 0, (5) the inverter model on B drives its output towards 1, (6) gate C presents a high resistance (low current) load, (7) the connection model sets its node to 1, (8) the inverter drives its output to 0, (9) gate A presents a low resistance (low current) load, and (10) the node model sets the node to 0 which contradicts the input of A being 1. An analogous analysis for the input of A being 0, yields a contradiction as well. The only remaining possible cardinality one diagnosis is T .

Using the temporal T models for the inverters produces a consistent analysis demonstrated in Table 1. This circuit is the familiar ring oscillator [17].

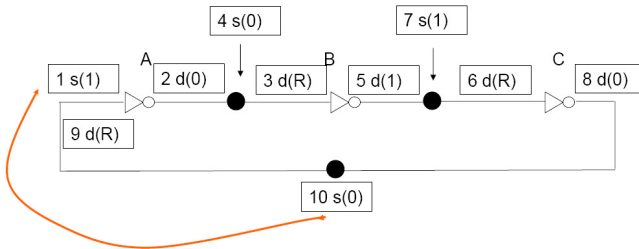


Fig. 7. Modeling connections does not remove the failure

Table 1. Outputs of the inverters of a ring oscillator after t gate delays. The oscillator takes 6 gate delays to return to its initial state, thus the output is a square wave with a period of 6 times the gate delay.

t	0	1	2	3	4	5	6
A	1	1	1	0	0	0	1
B	1	0	0	0	1	1	1
C	0	0	1	1	1	0	0

6.2 $\emptyset \rightarrow I$

Consider the Or – Xor circuit again (Fig. 6). For clarity assume the circuit has one fault. As derived in the Section 5.2, under the \emptyset models, Xor must be faulted. Suppose we measure the output of the Or gate at T_1 and T_2 to be 1 and then 0 respectively. In this case, we have derived the meta-conflict:

$$AB_a(T) \vee AB_a(C) \vee AB_a(I).$$

There are now three minimal candidate meta-diagnoses: T, C, I . The T meta-diagnosis immediately results in a failure yielding the meta-conflict:

$$AB_a(C) \vee AB_a(I).$$

The meta-diagnosis I yields a consistent point of view: Or is failing intermittently. The C meta-diagnosis cannot explain the observation:

$$AB_a(T) \vee AB_a(I).$$

6.3 $\emptyset \rightarrow C$

Consider the Or–Xor example again before the output of the Or gate is observed. Again, for clarity assume the circuit has one fault. Suppose Xor is replaced and the same symptoms reoccur. In this case, both the C and I meta-diagnoses are consistent. Under the I meta-diagnosis, the circuit contains two possible faults:

- Xor is faulted.
- Or is faulted.

The C meta-diagnosis is consistent, with 3 possible faults:

- The node at the output of Xor is shorted to power.
- The connection from the output Xor gate to the node is open and thus it floats to 1.
- The connection to $\text{in}_2(\text{Xor})$ is shorted to ground.

6.4 $\emptyset \rightarrow TCI$

Tasks which require a *TCI* preferred meta-diagnosis are complicated, but they do occur. Consider the four inverter system of Fig. 8. The input to inverter *Z* is held constant at 0. We assume single faults. Observing the output *D* is usually 0, but outputs 1s with no pattern. The observation $D = 1$ indicates that one of *Z*, *A*, *B*, *D* is faulted. However, a subsequent observation of $D = 0$ is inconsistent yielding the meta-conflict: $AB_a(T) \vee AB_a(C) \vee AB_a(I)$. No fault in the connections can produce the observations, therefore: $AB_a(T) \vee AB_a(I)$. No temporal fault can lead to this behavior either, so: $AB_a(I)$. Under meta-diagnosis *I*, the output of *A* is measured — it is usually 0, but sometimes 1. The output of *Z* is measured — it is usually 1, but sometimes 0. Therefore *Z* must be intermittently faulted (under meta-diagnosis *I*), but replacing it does not change the symptoms. This yields the meta-conflict: $AB_a(T) \vee AB_a(C)$. The *CI* meta-diagnosis also leads to an inconsistency. There is no fault within the connections that can explain the observations. Likewise there is no fault within the *TI* meta-diagnosis. The only meta-diagnosis that can explain the symptoms is *TCI*. The actual fault is an intermittent short between the output of *D* and output of *Z*. As the input to *Z* is 0, its output is 1. The connection models for digital gates are 0-dominant, so that, if a 0 from the output of *D* were feedback through an intermittent short, it would drive the input to *A* to 0. Thus for those times in which the intermittent short was manifest, the circuit would be a ring oscillator.

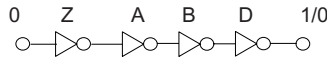


Fig. 8. A very simple circuit containing a very hard to pinpoint fault

7 Implementation

The analyses described in this paper have been implemented within the ATMS/HTMS framework [9,18]. Each domain or abstraction literal is represented by an explicit ATMS assumption in one ATMS instance. A fuller description of the *T*, *C*, and *I* abstractions can be found in [14,11,15,16]. The ATMS/HTMS architecture provides a unified framework to reason over any assumptions, be they about components or abstractions.

8 Related Work

Automated model abstraction has a long tradition in Artificial Intelligence. The graph of models ([7]) is similar to the meta-diagnosis lattice (Fig. 5) and analyzes conflicts to identify which modeling parameters to change. It is focused on design and analysis and the models that are constructed by hand. It does not use

diagnosis to guide the search for models, nor is it applied to diagnosis in some domain. Work on compositional modeling ([8]) also uses ATMS assumptions to represent domain abstractions and conflicts to guide the search for models. Again, the models are constructed by hand and do not use diagnosis at the domain or meta-levels. In context-dependent modeling ([19]) there is typically a much larger space of model fragments to choose from and explicit context information is used to guide the selection of the domain models. The task is to construct the best causal explanation for a physical phenomena. Yet again, the models are constructed by hand and do not use diagnosis at the domain or meta-levels.

In the model-based diagnosis literature, there has been considerable work on diagnostic assumptions and selecting appropriate models for a diagnostic task [12]. This paper focuses primarily on assumptions associated with choosing domain abstractions.

There has been considerable research on structural abstraction [5,6] where groups of components are combined to form larger systems to reduce computational complexity. [3] describes how the task can be used to partition the value of a variable into the qualitative values needed to solve a task. [4] presents another approach to partition the value of a variable into qualitative ranges to reduce complexity when there is limited observability of the variables.

9 Conclusions

This paper has presented a general approach to selecting the best domain abstraction level to address a task and has demonstrated it within the context of digital gates. In the case of digital gates the component models can be automatically generated from the basic models using domain schemas.

Acknowledgments. Daniel G. Bobrow and Elisabeth de Kleer provided many useful comments.

References

1. Struss, P.: What's in SD? Towards a theory of modeling for diagnosis. In: Console, L., (ed.) Working Notes of the 2st Int. Workshop on Principles of Diagnosis. Technical Report RT/DI/91-10-7, Dipartimento di Informatica, Università di Torino, Torino, Italy, 41–51 (1991)
2. Struss, P.: A theory of model simplification and abstraction for diagnosis. In: Proc. 5th Int. Workshop on Qualitative Physics, Austin, TX (1991)
3. Sachenbacher, M., Struss, P.: Task-dependent qualitative domain abstraction. *Artif. Intell.* 162(1-2), 121–143 (2005)
4. Torta, G., Torasso, P.: Automatic abstraction in component-based diagnosis driven by system observability. In: Gottlob, G., Walsh, T. (eds.) *IJCAI*, pp. 394–402. Morgan Kaufmann, San Francisco (2003)
5. Chittaro, L., Ranon, R.: Hierarchical model-based diagnosis based on structural abstraction. *Artif. Intell.* 155(1-2), 147–182 (2004)

6. Hamscher, W.C.: XDE: Diagnosing devices with hierarchic structure and known component failure modes. In: Proc. 6th IEEE Conf. on A.I. Applications, Santa Barbara, CA, pp. 48–54. IEEE Computer Society Press, Los Alamitos (1990)
7. Addanki, S., Cremonini, R., Penberthy, J.S.: Reasoning about assumptions in graphs of models. In: Proc. 11th Int. Joint Conf. on Artificial Intelligence, Detroit, MI, pp. 1432–1438 (1989)
8. Falkenhainer, B., Forbus, K.D.: Compositional modeling: Finding the right model for the job. *Artificial Intelligence* 51 95–143 (1991) Also In: de Kleer, J., Williams, B. (eds.) *Qualitative Reasoning about Physical Systems II*, pp. 95–143. North-Holland 1991/ MIT Press 1992, Amsterdam/Cambridge (1992)
9. de Kleer, J., Williams, B.C.: Diagnosing multiple faults. *Artificial Intelligence* 32 97–130 (1987) (Also). In: Ginsberg, M.L. (ed.) *Readings in NonMonotonic Reasoning*, pp. 280–297. Morgan Kaufmann, San Francisco (1987)
10. de Kleer, J., Mackworth, A., Reiter, R.: Characterizing diagnoses and systems. *Artificial Intelligence* 56(2-3), 197–222 (1992)
11. de Kleer, J.: Modeling when connections are the problem. In: Proc. 20th IJCAI, Hyderabad, India, pp. 311–317 (2007)
12. Böttcher, C.: No faults in structure? how to diagnose hidden interactions. In: IJCAI, pp. 1728–1735 (1995)
13. Böttcher, C., Dague, P., Taillibert, P.: Hidden interactions in analog circuits. In: Abu-Hakima, S. (ed.) *Working Papers of the Seventh International Workshop on Principles of Diagnosis*, Val Morin, Quebec, Canada, pp. 36–43 (1996)
14. Raiman, O., de Kleer, J., Saraswat, V., Shirley, M.H.: Characterizing non-intermittent faults. In: Proc. 9th National Conf. on Artificial Intelligence, Anaheim, CA, pp. 849–854 (1991)
15. de Kleer, J.: Diagnosing intermittent faults. In: 18th International Workshop on Principles of Diagnosis, Nashville, USA, pp. 45–51 (2007)
16. de Kleer, J.: Troubleshooting temporal behavior in combinational circuits. In: 18th International Workshop on Principles of Diagnosis, Nashville, USA pp. 52–58 (2007)
17. Wikipedia: Ring oscillator — Wikipedia, the free encyclopedia (2007) [Online; accessed 12-February-2007]
18. de Kleer, J.: A hybrid truth maintenance system. PARC Technical Report (January 1992)
19. Nayak, P.P.: *Automated Modeling of Physical Systems*. LNCS, vol. 1003, Springer, Heidelberg (1995)

Channeling Abstraction

Stijn De Saeger¹ and Atsushi Shimojima²

¹ School of Information Science, Japan Advanced Institute of Science and Technology
stijn@jaist.ac.jp*

² Faculty of Culture and Information Science, Doshisha University
ashimoji@mail.doshisha.ac.jp

Abstract. This paper presents work on the formalization of abstraction in knowledge representation. It is part of ongoing research on what might be termed “functional context dependence” or the relation between abstraction and idealization on the one hand, and approximation and the granularity of representation on the other. For the purpose of this paper we focus solely on abstraction. We briefly survey some existing theories, present an alternative framework for modeling abstraction based on channel theory and discuss two possible interpretations of the abstraction process — abstraction as a mapping between representations of different granularity, and abstraction as a theory.

1 Introduction: Vagueness and Granularity

“Two is company, three’s a crowd.”
(anonymous)

A large class of problems in philosophy and epistemology go under the header of “vagueness”. Specifically, giving a meaningful interpretation to vague terms like *tall*, *bald*, *smart*, *blue* and the like presents a real problem for classical first-order logic and its set theoretic semantics. To give an example, many if not most people are not clearly smart or clearly non-smart. The professed vagueness of these predicates is traditionally spelled out in terms of three related characteristics: (i) the existence of borderline cases, (ii) the absence of crisp boundaries for their applicability, and (iii) the fact that they all admit (some variation of) the *sorites* paradox. We refer the interested reader to [10] for a comprehensive overview of the literature on vagueness, and the *sorites* paradox in particular.

In the cognitive science and logical AI communities on the other hand, vagueness presents itself in a somewhat different guise. There the problem presented by vague descriptions is not so much an inherent inability to determine their precise extension, but the somewhat less exciting fact that people usually cannot be bothered to. Thus, the perceived vagueness of a proposition like “*Mount*

* Stijn De Saeger gratefully acknowledges support by the 21st Century COE program ‘Verifiable and Evolvable e-Society’ at JAIST. We thank the referees for their careful reviews and helpful comments.

Everest is about 8800m high” is not due to the fact that the exact height of Mount Everest is in principle indeterminate or unknowable, or that it is a matter of subjective opinion. More likely, it is due to implicit standards of precision that are in place in the context the utterance was made. In other words, this kind of vagueness is in a sense “functional”, and it can be restated in terms of *abstraction* and the level of *granularity* of a description.

1.1 Granularity and Knowledge Representation

The concept of granularity is closely related to the nature of knowledge representation itself. The formal representation of a given body of knowledge is characterized in [5] as the incremental process of determining an appropriate universe of discourse as domain, and formally interpreting its relevant entities in terms of some class of mathematical objects. Subsequently one defines a language for talking about the objects in this domain of discourse, together with a mapping from expressions in this language to the formal domain model. Exactly which aspects of the target domain are preserved and which are ignored in this process depends, among other things, on the purpose it serves us. The following quote by Davis et al. (1993, [2]) spells out this intuition.

“A knowledge representation is most fundamentally a *surrogate*, a substitute for the thing itself, used to enable an entity to determine consequences by thinking rather than acting, i.e. by reasoning about the world rather than taking action in it.” (Davis, [2])

By performing (mental) operations on a representation we can reason about the consequences of some specific state of affairs, action or event in the domain represented. To the extent that they lend themselves easily to such operations, knowledge representations exhibit obvious qualitative differences. For instance, a map or diagram is generally a more efficient representation than running text when attempting to follow some itinerary or reach some location. Any representation of potentially useful or relevant information then, whether it be sentences in some natural language or data in the knowledge base of some intelligent system, is a balancing act — it represents a trade-off between completer, more exhaustive descriptions of information on the one hand, and more economic but less qualified and potentially less general representations on the other. As such, abstraction is a tool for managing the complexity of a representation, and depending on its concrete use the same fact may be represented as widely different levels of abstraction.

In logical AI, *abstraction* has generated some interest as a mechanism of dynamically filtering out those aspects of a knowledge representation that are not immediately relevant to the problem at hand (see for instance [12,7,8,6,4]). Needless to say, any representation is an abstraction, in the sense that it represents some local optimum in a continuum of ever more detailed and fine-grained representations of the universe of discourse.

2 Abstraction as a Mapping

Throughout most of the literature ([12][78][4]), abstraction has predominantly been analyzed in terms of a mapping between two representations, i.e. from the base or “ground” representation to the abstract representation.

Giunchiglia and Walsh ([78]) define an abstraction $\Sigma_0 \Rightarrow \Sigma_1$ as a pair of formal systems $\Sigma_0 = \langle \mathcal{L}_0, \Delta_0, \Omega_0 \rangle$ and $\Sigma_1 = \langle \mathcal{L}_1, \Delta_1, \Omega_1 \rangle$, together with a function $f_{\mathcal{L}} : \mathcal{L}_0 \rightarrow \mathcal{L}_1$. Here \mathcal{L}_i is the formal language (set of well-formed formulae), Δ_i the deductive machinery (set of inference rules), and $\Omega_i \subset \mathcal{L}_i$ the set of axioms of Σ_i (for $i \in \{0, 1\}$). They add the further requirement that abstractions be *total*, *effective* and *surjective* functions. Total, because we should be able to apply the abstraction procedure to any given formula φ in the ground language and obtain its abstract version $f_{\mathcal{L}}(\varphi)$; effective because this whole process is useful only in so far as the abstraction $f_{\mathcal{L}}(\varphi)$ can actually be computed, and surjective because we want to use $f_{\mathcal{L}}$ to actually “generate” the abstract representation from the ground one.

A novel contribution of [78] is the analysis of abstractions in terms of which properties of the ground representation they preserve. Writing $TH(\Sigma_i)$ to denote the set of theorems derivable in a formal system Σ_i , they distinguish between the following cases:

Definition 1 (*T* abstractions*). *An abstraction $f : \Sigma_0 \Rightarrow \Sigma_1$ is a*

- (i) *TC abstraction when $\forall \varphi \in \mathcal{L}_0 : \varphi \in TH(\Sigma_0)$ iff $f_{\mathcal{L}}(\varphi) \in TH(\Sigma_1)$;*
- (ii) *TD abstraction when $\forall \varphi \in \mathcal{L}_0 : \text{if } f_{\mathcal{L}}(\varphi) \in TH(\Sigma_1) \text{ then } \varphi \in TH(\Sigma_0)$;*
- (iii) *TI abstraction when $\forall \varphi \in \mathcal{L}_0 : \text{if } \varphi \in TH(\Sigma_0) \text{ then } f_{\mathcal{L}}(\varphi) \in TH(\Sigma_1)$.*

So, any TC abstraction (“*Theorem Constant*”) is in fact both TD (“*Theorem Decreasing*”) and TI (“*Theorem Increasing*”). As Giunchiglia and Walsh investigate abstraction for its potential applications in theorem proving and automated proof search, their interest is mainly in TI abstractions, which serve as simplifications of some (possibly intractable) theory.

Nayak and Levy argue in ([12]) instead that abstractions are modeled more naturally by mappings between representations at the *semantic* level, rather than the syntactic level. One reason is that a naive translation approach may introduce unwarranted or even inconsistent conclusions in the abstract theory. The example they give concerns the following base theory ([12]):

$$\begin{aligned}
 \text{JapaneseCar}(x) &\Rightarrow \text{Car}(x) \\
 \text{EuropeanCar}(x) &\Rightarrow \text{Car}(x) \\
 \text{Toyota}(x) &\Rightarrow \text{JapaneseCar}(x) \\
 \text{BMW}(x) &\Rightarrow \text{EuropeanCar}(x)
 \end{aligned}
 \tag{1}$$

We can imagine reasoning tasks though for which the difference between Japanese and European cars is inconsequential, and in those cases we may want to abstract over them using a more general predicate `ForeignCar`, giving the simplified knowledge base in [2]

$$\begin{aligned}
\text{ForeignCar}(x) &\Rightarrow \text{Car}(x) \\
\text{Toyota}(x) &\Rightarrow \text{ForeignCar}(x) \\
\text{BMW}(x) &\Rightarrow \text{ForeignCar}(x)
\end{aligned} \tag{2}$$

If however the original knowledge base contained additional axioms like the ones below, the abstraction process is no longer sound:

$$\begin{aligned}
\text{EuropeanCar}(x) &\Rightarrow \text{Fast}(x) \\
\text{JapaneseCar}(x) &\Rightarrow \text{Reliable}(x)
\end{aligned}$$

This would now allow one to derive, among other things, that $\text{ForeignCar}(x) \Rightarrow \text{Fast}(x)$ and therefore also $\text{Toyota}(x) \Rightarrow \text{Fast}(x)$, a conclusion not warranted by the base theory. As explained in [12], the reason such anomalies occur becomes clear when recalling the modus operandi of formal knowledge representation. One *first* captures the entities of interest in some target domain in a class of appropriate mathematical objects (sets, relations, . . .), over which a formal language is then defined inductively as a set of logical formulae. Abstraction as a mere *syntax* level mapping between formal languages is prone to “alignment errors”, not with respect to the actual translation but with respect to the intended target domain, as is the case above.

Instead, Nayak and Levy propose to treat abstractions as model mappings. If \mathcal{L}_{base} and \mathcal{L}_{abs} are the base language and abstraction language respectively, then abstractions are defined in [12] as functions π from (sets of) interpretations of \mathcal{L}_{base} to (sets of) interpretations of \mathcal{L}_{abs} :

$$\pi : \text{Interpretations}(\mathcal{L}_{base}) \rightarrow \text{Interpretations}(\mathcal{L}_{abs}) \tag{3}$$

Given a theory (i.e. set of formulae) T_{base} in the ground language then, a theory T_{abs} is the abstract image of T_{base} under π if and only if for every model $I \in \text{Interpretations}(\mathcal{L}_{base})$ that is a model of T_{base} , its image $\pi(I) \in \text{Interpretations}(\mathcal{L}_{abs})$ is a model of T_{abs} .

Formally, an interpretation mapping π is defined in terms of an auxiliary function f_π relating formulae $\varphi' \in \mathcal{L}_{abs}$ to formulae $\varphi \in \mathcal{L}_{base}$, such that the denotation of φ' can be defined as $\pi(\text{Interpretations}(f_\pi(\varphi')))$. The abstraction then consists of a wff π_\forall with one free variable (whose extension is used to define the abstract domain), and for each n -ary predicate P in \mathcal{L}_{abs} a wff π_P with n free variables such that, for any $I \in \text{Interpretations}(T_{base})$, π_P denotes an n -ary relation in $\text{dom}(I)$. Then P 's denotation in $\pi(I)$ is just this relation restricted to the domain of π_\forall . For instance, in the car theory in example 1, π preserves the domain of discourse (just take any tautology for π_\forall). $f_\pi(\text{ForeignCar})$ becomes the wff $\text{JapaneseCar}(x) \vee \text{EuropeanCar}(x)$, and all other predicates (except JapaneseCar and EuropeanCar , who are not in \mathcal{L}_{abs}) are simply mapped to themselves. Furthermore, f_π is defined inductively on the structure of composite formulae, and $f_\pi(\forall x. \varphi)$ becomes $\forall x. \pi_\forall(x) \Rightarrow f_\pi(\varphi)$ in order to enforce the domain restriction (π_\forall) associated with the abstraction.

Nayak and Levy call these mappings *Model Increasing* (MI) abstractions, in analogy with Giunchiglia and Walsh's classification. In fact, given Nayak and Levy's

requirement for abstractions that preserve the non-validity of theorems across π mappings, MI abstractions correspond to a strict subset of TD abstractions.

To summarize, both theories conceive of abstraction as a mapping process between different formal representations. Giunchiglia and Walsh define this mapping at the syntactic level: given formal systems Σ_0 and Σ_1 , an abstraction is a function mapping \mathcal{L}_0 onto \mathcal{L}_1 , thereby fully generating the abstract representation from the ground one. The advantage of this approach is clear: abstractions can be generated mechanically, possibly even at run-time. The flip side of this coin is that such mechanical process may introduce theorems not licensed by the original base theory. Often this effect is intended; Giunchiglia and Walsh are interested in abstractions as computationally tractable reformulations of a given base theory, under certain simplifying assumptions. As Nayak and Levy’s example demonstrates though, without a corresponding semantic interpretation, the effects of these so-called “simplifying assumptions” are difficult to contain or control. Instead, Nayak and Levy propose a mapping at the level of domain interpretations. While this approach is able to keep close tabs on the soundness of the resulting abstract theory, it does so at the cost of introducing several additional levels of indirection in the abstraction process.

Recently Giunchiglia and Ghidini ([4]) have proposed a promising theory of abstraction grounded in local models semantics ([3]). Here again abstraction is modeled as a syntax-level mapping much as in [78] but given a rigorous semantics in terms of *domain relations*, i.e. compatibility constraints between the domains of the respective ground and abstract representations. While technically local models semantics is rather different from the formal model we are about to introduce in the following sections, we believe many of the underlying assumptions and ideas to be compatible, and plan to investigate the respective strengths and weaknesses of both frameworks in future research.

3 Abstraction in Channel Theory

Analogous to the discussion in the previous section, we treat abstraction as a mapping between formal representations for now. In the next section, we will explore an alternative approach of modeling abstractions as theories in their own right.

Channel theory (Barwise and Seligman, [1]) was conceived as a mathematical framework for analyzing the flow of information in distributed systems. In contrast to Shannon’s information theory though, channel theory is essentially a *qualitative* theory of information — rather than raw quantity (in bits), it is concerned with *what kind* of information may flow between components in a given distributed system. The components themselves are represented through objects called *classifications*.

Definition 2 (Classification). *A classification \mathcal{C} is a triple $\langle \text{typ}(\mathcal{C}), \text{tok}(\mathcal{C}), \models \rangle$, where $\text{typ}(\mathcal{C})$ and $\text{tok}(\mathcal{C})$ are sets respectively called the types and tokens of \mathcal{C} , and $\models \subseteq \text{tok}(\mathcal{C}) \times \text{typ}(\mathcal{C})$ is a binary classification relation.*

At the heart, classifications express a very general and indeed simple idea. Formal knowledge representations, and logics in particular, are special cases of classifications $\langle \mathcal{L}, M, \models \rangle$, where \mathcal{L} is a formal language (a set of formulae), M is a set of semantic structures (models in which to evaluate formulae of \mathcal{L}), and \models is the usual satisfaction relation, defined inductively on the structure of formulae $\varphi \in \mathcal{L}$. Note that characterizing logics in terms of classifications does not commit us to any specific outlook of the logic — \mathcal{L} can be a set of attributes closed under propositional connectives, in which case we take M as a set of truth assignments to the atoms of \mathcal{L} . In case \mathcal{L} also contains modal operators, M may be a Kripke frame, i.e. a set of worlds W with some internal structure \prec defined on it. Or \mathcal{L} can be a first-order language, in which case M is a class of first-order structures $\langle \text{dom}, \mathcal{I} \rangle$, and so forth.

By abuse of notation, let's take $\Gamma \models_m \Delta$ (for some $m \in M$) to mean that when $m \models \gamma$ for all formulae $\gamma \in \Gamma$, then $m \models \delta$ for some formula $\delta \in \Delta$. Furthermore, we write $\Gamma \models_X \Delta$ when $\Gamma \models_m \Delta$ for all $m \in X \subset M$, and $\Gamma \models \Delta$ proper when $X = M$. As a general result then, any classification relation \models satisfies the following:¹

Identity:	$\varphi \models \varphi$	(for $\varphi \in \mathcal{L}$)
Weakening:	if $\Gamma \models \Delta$ then $\Gamma, \Gamma' \models \Delta, \Delta'$	(for $\Gamma, \Gamma', \Delta, \Delta' \subseteq \mathcal{L}$)
Global Cut:	if $\Gamma, \Sigma_0 \models \Delta, \Sigma_1$ for each partition $\langle \Sigma_0, \Sigma_1 \rangle$ of Σ , then $\Gamma \models \Delta$	(for $\Gamma, \Delta, \Sigma \subseteq \mathcal{L}$)

Generalizing, any consequence relation \vdash on \mathcal{L} is said to be *regular* when it satisfies these three structural rules of Identity, Weakening and Global Cut above. The resemblance to Gentzen-style sequent notation is of course intended. The idea of a consequence relation on the types of a classification \mathcal{C} gives rise to the notion of a *local logic*.

Definition 3 (Local logic). A local logic L is a triple $\langle \mathcal{C}, \vdash, N \rangle$, consisting of a classification \mathcal{C} , a consequence relation $\vdash \subseteq \text{typ}(\mathcal{C}) \times \text{typ}(\mathcal{C})$, and a set $N \subseteq \text{tok}(\mathcal{C})$ called the “normal models” of L .

The normal models in N represent the set of situations the theory of L is intended to capture, i.e. “is about”. For our current purpose we only consider consistent theories, so we require that $\Gamma \models_N \Delta$ whenever $\Gamma \vdash \Delta$. In case $N = \text{tok}(\mathcal{C})$, we say the logic L is *sound*. Dually, L is *complete* iff $\Gamma \vdash \Delta$ whenever $\Gamma \models_N \Delta$, and *globally complete* when $N = \text{tok}(\mathcal{C})$. We now try to recast Nayak and Levy’s car theory in this setting.

Example 1. Let \mathcal{C}_B be a (base) classification $\langle \mathcal{L}_B, M_B, \models_B \rangle$. Nayak and Levy’s example in [2] does not seem to require a full first-order theory, so we just define \mathcal{L}_B as a formal language built from a set of variables $\text{Var} = \{x, y, z\}$ (possibly subscripted) and a set of (unary) predicates $\text{Pred} = \{\text{Car}, \text{JapaneseCar}, \text{EuropeanCar}, \text{Fast}, \text{Reliable}, \text{Toyota}, \text{BMW}\}$ closed under the usual connectives.

Let M_B be a set of first-order structures $\langle \text{dom}, \mathcal{I} \rangle$, where dom is a non-empty set. Unless stated otherwise, we take the domain of discourse dom to be fixed for

¹ See [1], p.119.

all members of $M_{\mathbf{B}}$ and thus refer to structures simply by their interpretation function I . Further simplifying the standard first-order semantics, we will treat I straightforwardly as a function assigning an element $\llbracket v \rrbracket^I \in \text{dom}$ to each variable v in Var , and subsets $\llbracket P \rrbracket^I$ of dom to the predicates. Then the classification relation $\models_{\mathbf{B}}$ is defined on the structure of formulae as:

$$\begin{aligned} I \models_{\mathbf{B}} P(v) &\text{ iff } \llbracket v \rrbracket^I \in \llbracket P \rrbracket^I \text{ (for } P \in \text{Pred}, v \in \text{Var}). \\ I \models_{\mathbf{B}} \neg\varphi &\text{ iff } I \not\models_{\mathbf{B}} \varphi. \\ I \models_{\mathbf{B}} \varphi \wedge \psi &\text{ iff } I \models_{\mathbf{B}} \varphi \text{ and } I \models_{\mathbf{B}} \psi. \end{aligned}$$

Further connectives \vee , \Rightarrow and \Leftrightarrow are given as the usual abbreviations. Then the ground theory given in [2](#) can be represented as a local logic $L_{\mathbf{B}} = \langle \mathcal{C}_{\mathbf{B}}, \vdash_{\mathbf{B}}, N_{\mathbf{B}} \rangle$, where $\vdash_{\mathbf{B}}$ is defined as the regular closure of the following theory:

$$\begin{aligned} \text{JapaneseCar}(x) &\vdash \text{Car}(x) \\ \text{EuropeanCar}(x) &\vdash \text{Car}(x) \\ \text{Toyota}(x) &\vdash \text{JapaneseCar}(x) \\ \text{BMW}(x) &\vdash \text{EuropeanCar}(x) \end{aligned} \tag{4}$$

Its normal models $N_{\mathbf{B}}$ contains all models $I \in M_{\mathbf{B}}$ such that $I \models_{\mathbf{B}} \bigwedge \Gamma \Rightarrow \bigvee \Delta$ for all constraints $\langle \Gamma, \Delta \rangle$ in $\vdash_{\mathbf{B}}$.

The corresponding abstract classification $\mathcal{C}_{\mathbf{A}}$ is defined analogously, as a triple $\langle \mathcal{L}_{\mathbf{A}}, M_{\mathbf{A}}, \models_{\mathbf{A}} \rangle$. Here we take $\mathcal{L}_{\mathbf{A}}$ to be similar to $\mathcal{L}_{\mathbf{B}}$, except for consisting of predicates $\{\text{Car}, \text{ForeignCar}, \text{Fast}, \text{Reliable}, \text{Toyota}, \text{BMW}\}$. The set of models $M_{\mathbf{A}}$ for this language and the corresponding classification relation $\models_{\mathbf{A}}$ looks as can be expected from the definition of $\mathcal{C}_{\mathbf{B}}$, modulo differences in their respective languages.

In order to set up the abstraction mapping between $\mathcal{C}_{\mathbf{B}}$ and $\mathcal{C}_{\mathbf{A}}$, we need to introduce some further machinery.

Definition 4 (Infomorphism). *Given classifications $\mathcal{C} = \langle \text{typ}(\mathcal{C}), \text{tok}(\mathcal{C}), \models_{\mathcal{C}} \rangle$ and $\mathcal{C}' = \langle \text{typ}(\mathcal{C}'), \text{tok}(\mathcal{C}'), \models_{\mathcal{C}'} \rangle$, an infomorphism $f : \mathcal{C} \rightleftarrows \mathcal{C}'$ from \mathcal{C} to \mathcal{C}' is a pair of functions $\langle f^{\wedge}, f^{\vee} \rangle$ satisfying the following biconditional:*

$$\forall \sigma \in \text{typ}(\mathcal{C}), s \in \text{tok}(\mathcal{C}') : f^{\vee}(s) \models_{\mathcal{C}} \sigma \text{ iff } s \models_{\mathcal{C}'} f^{\wedge}(\sigma)$$

A more visual way of expressing the infomorphism condition is to require that the diagram in Fig. [11](#) commutes (note the different directions of functions f^{\wedge} and f^{\vee}).

Infomorphisms formalize a basic correspondence in the information structure of two separate classifications — they state that the way these classifications capture regularities in the target domain is fundamentally compatible. Additionally, infomorphisms allow to move sequents and indeed entire local logics back and forth across classifications, precisely keeping track of when and where soundness and/or completeness of the logic in question is compromised. To see how this works in practice we turn back to our example, involving the car classifications $\mathcal{C}_{\mathbf{B}}$ and $\mathcal{C}_{\mathbf{A}}$.

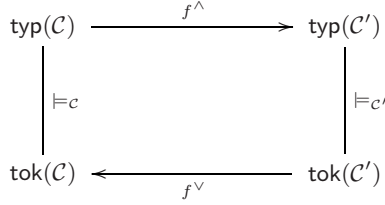


Fig. 1. Infomorphism $f : \mathcal{C} \rightleftharpoons \mathcal{C}'$

Example 2. Let $f : \mathcal{C}_B \rightleftharpoons \mathcal{C}_A$ be an infomorphism between classifications \mathcal{C}_B and \mathcal{C}_A , as defined in example 1. Concretely, let f consist of the following functions on tokens and types.

$$(i) \ f^\wedge(\varphi) : \mathcal{L}_B \rightarrow \mathcal{L}_A = \begin{cases} \neg f^\wedge(\psi) & \text{if } \varphi = \neg\psi \\ f^\wedge(\psi) \otimes f^\wedge(\psi') & \text{if } \varphi = \psi \otimes \psi' \text{ and } \otimes \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\} \\ \text{ForeignCar}(v) & \text{if } \varphi = \text{EuropeanCar}(v) \text{ (for } v \in \text{Var}) \\ \text{ForeignCar}(v) & \text{if } \varphi = \text{JapaneseCar}(v) \text{ (for } v \in \text{Var}) \\ \varphi & \text{otherwise} \end{cases}$$

$$(ii) \ f^\vee(\langle \text{dom}, I \rangle) : M_A \rightarrow M_B = \langle \text{dom}', I' \rangle \text{ where } \text{dom}' = \text{dom} \text{ and } I' = I \text{ except } \llbracket \text{JapaneseCar} \rrbracket^{I'} = \llbracket \text{EuropeanCar} \rrbracket^{I'} = \llbracket \text{ForeignCar} \rrbracket^I.$$

Here f^\wedge is the language-level abstraction function. In fact, it can easily be seen to correspond to so-called *atomic abstractions* in the terminology of [4], as the abstraction mapping is defined on the atoms of the language while preserving the structure of formulae. As discussed in [4], this kind of abstraction between languages represents a sort of ideal-case scenario. In the next section we will have a look at some less straightforward cases, where the abstraction is not merely a function but rather a theory on the alignment of two classifications, which can itself subsequently be reasoned about — a form of “higher-order” abstraction, if you will.

If f^\wedge is a syntax-level mapping, f^\vee is its dual defined at the level of *models*. Note that its definition closely mirrors that of f^\wedge , be it at the semantic level. To verify that f indeed satisfies the infomorphism condition, consider a formula $\text{EuropeanCar}(z) \in \mathcal{L}_B$ and an arbitrary model $I \in M_A$. If $I \vDash_A \text{EuropeanCar}(z)$ (i.e. $I \vDash_A \text{ForeignCar}(z)$) then $\llbracket z \rrbracket^I \in \llbracket \text{ForeignCar} \rrbracket^I$ and thus by virtue of the definition of f^\vee , $\llbracket z \rrbracket^{f^\vee(I)}$ is a member of $\llbracket \text{EuropeanCar} \rrbracket^{f^\vee(I)}$. It is straightforward to show that the same reasoning applies to any arbitrary formula φ in \mathcal{L}_B .²

As we hinted at above, infomorphisms interact with local logics in interesting ways. By extension, the image of a sequent $\langle \Gamma, \Delta \rangle$ under f^\wedge is $\langle f^\wedge(\Gamma), f^\wedge(\Delta) \rangle$. Transporting sequents back and forth between classifications over infomorphisms

² Exercise for the reader: what happens in the case of a formula like $\text{EuropeanCar}(x) \wedge \neg \text{JapaneseCar}(x)$?

then can take the form of an inference rule. Formally, given $f : \mathcal{C} \rightleftarrows \mathcal{C}'$, and two logics L and L' on \mathcal{C} and \mathcal{C}' respectively, define two associated inference rules:

$$\frac{\Gamma \vdash_L \Delta}{f^\wedge(\Gamma) \vdash_{L'} f^\wedge(\Delta)} f\text{-Intro} \qquad \frac{f^\wedge(\Gamma) \vdash_{L'} f^\wedge(\Delta)}{\Gamma \vdash_L \Delta} f\text{-Elim} \quad (5)$$

This transfer of reasoning between classifications over an interpretation function f extends all the way up to the concept of local logics — the image of a local logic $L = \langle \mathcal{C}, \vdash, N \rangle$ under f (written $f[L]$) is defined as $L' = \langle \mathcal{C}', \vdash', N' \rangle$ where:

- (i) $\vdash' = \{ \langle f^\wedge(\Gamma), f^\wedge(\Delta) \rangle \mid \Gamma \vdash \Delta \}$
- (ii) $N' = \{ s \in \text{tok}(\mathcal{C}') \mid f^\vee(s) \in N \}$

Similarly, the inverse image of L' (written $f^{-1}[L']$) has the following theory \vdash and normal models N :

- (i) $\vdash = \{ \langle \Gamma, \Delta \rangle \mid f^\wedge(\Gamma) \vdash' f^\wedge(\Delta) \}$
- (ii) $N = \{ f^\vee(s) \in \text{tok}(\mathcal{C}) \mid s \in N' \}$

Thus, the inference rules f -Intro and f -Elim allow for reasoning to switch between the base representation to a more abstract representation of some domain. As we have seen, this process of abstraction sometimes introduces errors and inaccuracies, either intended or not. It is important therefore to have a firm handle on the specific conditions under which these errors arise. Recalling some general results from [1], consider again two arbitrary local logics L and L' on \mathcal{C} and \mathcal{C}' , and let $f : \mathcal{C} \rightleftarrows \mathcal{C}'$. Due to the infomorphism condition, an application of f -Intro can be seen to preserve soundness in case $f^\vee(N') \subseteq N$, and completeness (i.e. non-validity of inferences) when $f^\vee(N') \supseteq N$. Dually, using f -Elim to reason *against* the direction of the abstraction is sound when $N \subseteq f^\vee(N')$, and complete when $N \supseteq f^\vee(N')$.

Example 3. Returning to the car theory example, and the local logic L_B representing the original ground theory depicted in (4). Its associated abstract local logic $L_A = f[L_B]$ corresponds to Nayak and Levy's abstract theory given in (2).

$$\begin{aligned} \text{JapaneseCar}(x) \vdash_B \text{Car}(x) &\xrightarrow{f^\wedge} \text{ForeignCar}(x) \vdash_A \text{Car}(x) \\ \text{EuropeanCar}(x) \vdash_B \text{Car}(x) &\xrightarrow{f^\wedge} \text{ForeignCar}(x) \vdash_A \text{Car}(x) \\ \text{Toyota}(x) \vdash_B \text{JapaneseCar}(x) &\xrightarrow{f^\wedge} \text{Toyota}(x) \vdash_A \text{ForeignCar}(x) \\ \text{BMW}(x) \vdash_B \text{EuropeanCar}(x) &\xrightarrow{f^\wedge} \text{BMW}(x) \vdash_A \text{ForeignCar}(x) \end{aligned} \quad (6)$$

In general, L_B does not hold for the entire class of models M_B ; the set of normal models N_B represents those situations for which the theory was intended. Because of the above discussion then, we know L_A is at least as sound as L_B — N_A contains only models whose image under f^\vee are members of N_B , and therefore $f^\vee(N_A) \subseteq N_B$ holds trivially.

In contrast, abstraction over f -Intro is known to preserve non-validity only in case $f^\vee(N_A) \supseteq N_B$, as f^\vee being surjective on N_B would guarantee that any possible counterexample model $I \in N_B$ to some invalid sequent $\Gamma \not\vdash_B \Delta$ is the image of some counterexample $f^{\wedge^{-1}}(I) \in N_A$. In our example, f^\vee is clearly *not* surjective on M_B — the only ground interpretations I in its range are those in which $\llbracket \text{JapaneseCar} \rrbracket^I$ is equal to $\llbracket \text{EuropeanCar} \rrbracket^I$. And so an invalid sequent $\text{BMW}(y) \not\vdash_B \text{JapaneseCar}(y)$ in L_B becomes valid when abstracted to $\text{BMW}(y) \vdash_A \text{ForeignCar}(y)$. Similarly, as Nayak and Levy’s analysis in [12] points out, if the base theory were to contain further information like $\text{EuropeanCar}(x) \vdash_B \text{Fast}(x)$, the regular closure (see [3]) of $f[L_B]$ augmented with this specific sequent would indeed yield unwarranted inferences like the one below.

$$\frac{\frac{\text{EuropeanCar}(x) \vdash_B \text{Fast}(x)}{\text{ForeignCar}(x) \vdash_A \text{Fast}(x)} \text{ } f\text{-Intro} \quad \frac{\text{Toyota}(x) \vdash_B \text{JapaneseCar}(x)}{\text{Toyota}(x) \vdash_A \text{ForeignCar}(x)} \text{ } f\text{-Intro}}{\text{Toyota}(x) \vdash_A \text{Fast}(x)} \text{GC}}$$

Indeed, in terms of Giunchiglia and Walsh’s characterization in [7,8], abstraction tends to be a theorem-increasing process, at least in the general case. Among other things, this connects the notion of abstraction with that of *idealization*, as discussed by Hobbs in [9]. An abstraction serves as an idealization when it knowingly introduces unsound inferences for the sake of simplifying certain regularities in the target domain.

In fact, by virtue of the infomorphic relation between the base and abstract representation, the abstraction process can act as an important source of feedback concerning possible “holes” in the base theory as well. As argued in [11], local logics often behave as *contexts*, in the sense that they encode implicit assumptions and background conditions. Thought of as contexts, local logics consist of a logical theory and a set of models or situations taken to be “normal” with respect to this theory. *Strengthening* a local logic by adding to its theory is therefore offset by a decrease in normal models, given the requirement that $\Gamma \models_{N'} \Delta$ when $\Gamma \vdash' \Delta$. For two logics $L = \langle \mathcal{C}, \vdash, N \rangle$ and $L' = \langle \mathcal{C}, \vdash', N' \rangle$, L' is a *stronger* logic than L (written $L \sqsubseteq L'$) when $\vdash \subseteq \vdash'$ and $N \supseteq N'$. This natural order \sqsubseteq on the set of local logics on \mathcal{C} is known to form a complete lattice, with meet and join operations as given below ([11, 12.26])³

$$\begin{aligned} L \sqcap L' &=_{\text{def}} \langle \mathcal{C}, \text{Reg}(\vdash \cap \vdash'), N \cup N' \rangle \\ L \sqcup L' &=_{\text{def}} \langle \mathcal{C}, \text{Reg}(\vdash \cup \vdash'), N \cap N' \rangle \end{aligned}$$

This has interesting implications for the abstraction process, and our car example in particular. Recall the abstract car theory $L_A = f[L_B]$. The set of normal models N_A of L_A is not empty; however, *every* model $I \in N_A$ that is normal with respect to L_A is one whose interpretation as a normal model of the base theory $f^\vee(I)$ *collapses* the extents of predicates *JapaneseCar* and *EuropeanCar*. This

³ Here $\text{Reg}(S)$ is the regular closure of a set of sequents S , as introduced in [3].

is possible because at no point did we specify anything about the relation between predicates `EuropeanCar` and `JapaneseCar`. Intuitively though, we can easily imagine being interested instead in the stronger base logic $L'_B \supseteq L_B$ in which `EuropeanCar` and `JapaneseCar` are in fact *disjoint* concepts, that is, the local logic whose normal models $I \in N'_B$ all have in common that $\llbracket \text{EuropeanCar} \rrbracket^I \cap \llbracket \text{JapaneseCar} \rrbracket^I = \emptyset$. Let L'_B be the local logic obtained by extending the original theory in L_B with a sequent “ $\langle \{ \text{JapaneseCar}(x), \text{EuropeanCar}(x) \}, \emptyset \rangle$ ” (i.e. $\neg(\text{JapaneseCar}(x) \wedge \text{EuropeanCar}(x))$) and computing its regular closure. Now, the corresponding abstract theory $L'_A = f[L'_B]$ is no longer satisfiable: given that no model $I \in M_A$ has an image that is a normal model of L'_B , $N'_A = \emptyset$.

4 Abstraction as a Theory

As we saw in the previous section, mapping based abstraction methods provide an elegant solution in those cases where the abstraction in question can be defined straightforwardly on the atoms of the ground representation. Sometimes though, matters are not so simple. Or rather, as in the example above, sometimes we wish to be able to reason explicitly about specific background conditions under which we deem the abstraction process appropriate. Intuitively, the idea is to represent and ultimately reason with the abstraction itself as a theory on the alignment of two representations — the ground and the abstract one. As we already have a way of representing theories (i.e. local logics), the formal machinery introduced so far will suffice. We just introduce some further terminology.

Definition 5 (Channel). *A channel is a classification that serves to connect other classifications. Formally, a channel is a tuple $\langle \mathcal{C}, \{f_i : \mathcal{C}_i \rightleftharpoons \mathcal{C}\}_{i \in I} \rangle$ consisting of an indexed family of infomorphisms f_i (for some index I) with a common co-domain classification \mathcal{C} , called the channel “core”.*

For our purposes, we will restrict the discussion to *binary* channels, which are classifications connecting pairs of classifications. Then, instead of thinking of abstractions as a mapping between classifications, we now think of abstractions as local logics on a channel core \mathcal{C}_C , to which the respective base and abstract theories have been lifted via infomorphisms $f : \mathcal{C}_B \rightleftharpoons \mathcal{C}_C$ and $g : \mathcal{C}_A \rightleftharpoons \mathcal{C}_C$.

While there are many ways to set up such a construction, the “canonical” way is to define \mathcal{C}_C simply as the sum classification $\mathcal{C}_B + \mathcal{C}_A$ (see [11], p.81). In the interest of being able to compute the abstract theory generically from the ground theory, this seems like a good way to proceed. Let $\mathcal{C}_C = \mathcal{C}_B + \mathcal{C}_A$ be a classification $\langle M_C, \mathcal{L}_C, \models_C \rangle$, where \mathcal{L}_C is the disjoint union of \mathcal{L}_B and \mathcal{L}_A , in which each atomic proposition is indexed according to the original language it was taken from. Thus, \mathcal{L}_C will contain both predicates Fast_B and Fast_A , where Fast_B corresponds to $\text{Fast} \in \mathcal{L}_B$ and Fast_A is the respective version from \mathcal{L}_A . At the semantic level, members of M_C are pairs of models $\langle I, I' \rangle$ (for $I \in M_B, I' \in M_A$) — that is,

M_C is the Cartesian product $M_B \times M_A$. Then the classification relation \models_C is defined compositionally for $\langle I, I' \rangle \in M_C$ as:

$$\begin{aligned} \langle I, I' \rangle \models_C \varphi_B &\text{ iff } I \models_B \varphi \\ \langle I, I' \rangle \models_C \varphi_A &\text{ iff } I' \models_A \varphi \end{aligned}$$

Together with infomorphisms $f : \mathcal{C}_B \rightleftarrows \mathcal{C}_C$ and $g : \mathcal{C}_A \rightleftarrows \mathcal{C}_C$, something like the picture in Fig. 2 emerges. Both f^\wedge and g^\wedge map formulae φ to their indexed version in \mathcal{L}_C , and f^\vee and g^\vee act simply as accessor functions, returning respectively the left or right component model of a pair $\langle I, I' \rangle \in M_C$.

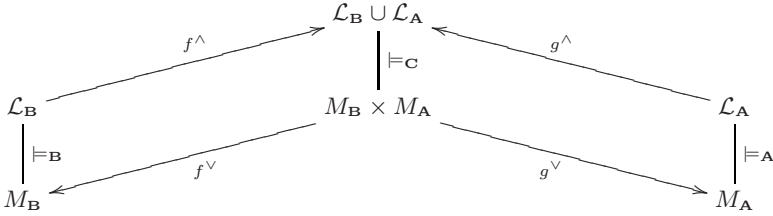


Fig. 2. A binary channel $\mathcal{C}_C = \mathcal{C}_B + \mathcal{C}_A$

In this setting, we can now represent the reasoning behind the abstraction process itself as a logical theory on the alignment of \mathcal{C}_B and \mathcal{C}_A . Let $L_C = \langle \mathcal{C}_C, \vdash_C, N_C \rangle$ be such a local logic. The theory \vdash_C of L_C might initially contain the following constraints (again, closed under regularity).

$$\begin{aligned} \text{Fast}_B(x) &\vdash_C \text{Fast}_A(x) \\ \text{Fast}_A(x) &\vdash_C \text{Fast}_B(x) \\ \text{Car}_B(x) &\vdash_C \text{Car}_A(x) \\ \text{Car}_A(x) &\vdash_C \text{Car}_B(x) \\ &\vdots \\ \text{EuropeanCar}_B(x) &\vdash_C \text{ForeignCar}_A(x) \\ \text{JapaneseCar}_B(x) &\vdash_C \text{ForeignCar}_A(x) \\ \text{ForeignCar}_A(x) &\vdash_C \text{JapaneseCar}_B(x), \text{EuropeanCar}_B(x) \end{aligned} \tag{7}$$

Note that as a logical theory, \vdash_C now *translates* between \mathcal{C}_B and \mathcal{C}_A in both directions, mapping formulae from \mathcal{L}_B to \mathcal{L}_A and vice versa. In other words, abstraction now behaves as a bi-directional mapping between \mathcal{L}_B and \mathcal{L}_A . The normal models of N_C consists of those pairs of models $\langle I, I' \rangle$ that satisfy the theory above, that is, all combinations of base models $I \in M_B$ and abstract models $I' \in M_A$ that are properly “aligned” according to the theory of L_C .

So given our original base theory L_B , we can compute its corresponding abstract theory by lifting L_B into \mathcal{C}_C over f , computing the implications of joining

$f[L_B]$ with the abstraction logic L_C , and finally exporting the resulting theory to \mathcal{C}_A via the inverse of g .

$$L_A = g^{-1}[f[L_B] \sqcup L_C] \tag{8}$$

Furthermore, because both f^\vee and g^\vee are surjective on M_B and M_A by definition, applications of the inference rules f -Intro, f -Elim, g -Intro and g -Elim are all guaranteed to be sound and complete. A little example of reasoning with abstraction theories is given in Fig. 3.

$$\frac{\frac{\frac{BMW(x) \vdash_B \text{EuropeanCar}(x)}{BMW_B(x) \vdash_{f[L_B]} \text{EuropeanCar}_B(x)}{BMW_B(x) \vdash_{f[L_B] \sqcup L_C} \text{ForeignCar}_A(x)} \quad \frac{\text{EuropeanCar}_B(x) \vdash_C \text{ForeignCar}_A(x)}{BMW_A(x) \vdash_C \text{BMW}_B(x)} \text{GC}}{BMW_A(x) \vdash_{f[L_B] \sqcup L_C} \text{ForeignCar}_A(x)} \text{GC}}{BMW(x) \vdash_{g^{-1}[f[L_B] \sqcup L_C]} \text{ForeignCar}(x)} \text{g-Elim}$$

Fig. 3. Reasoning with abstractions as theories

As the theory in \vdash_C can contain sequents of arbitrary complexity, representing the abstraction itself as a local logic gives a genuinely more expressive way of modeling the abstraction process than an atomic level mapping between languages. In addition, because they are local logics we can now strengthen or weaken the abstraction itself, depending on whether the concrete situations we encounter can be regarded as “normal” with respect to the *intent* of the abstraction process. For example, we may wish to define a second local logic $L'_C \sqsupseteq L_C$ on \mathcal{C}_C , representing a stronger abstraction theory than L_C . This logic L'_C can contain additional constraints to the ones in L_C , say for instance:

$$\begin{aligned} \text{EuropeanCar}_B(x) \vdash'_C \neg \text{JapaneseCar}_B(x) \\ \text{JapaneseCar}_B(x) \vdash'_C \neg \text{EuropeanCar}_B(x) \end{aligned} \tag{9}$$

The normal models of L'_C will then be a strict subset of those in L_C . Specifically, it will exclude undesirable combinations $\langle I, I' \rangle$ where $\llbracket \text{EuropeanCar}_B \rrbracket^{\langle I, I' \rangle} \cap \llbracket \text{JapaneseCar}_B \rrbracket^{\langle I, I' \rangle} \neq \emptyset$ as abnormal.

5 Discussion

In this paper we have sketched the beginnings of a theory of abstraction based on the notion of information flow introduced in [11]. As one referee correctly pointed out, it is no silver bullet — it still does not generate the abstract representation for us, and based on the data it is given to work with, the model still does not tell us that Japanese cars cannot be European cars. Indeed, short of providing it with a set of real data to learn from, we don't see how it could. However, if desirable such *observation-based* logics could be incorporated in the current

model rather easily: just add a new classification (\mathcal{C}_{cars} , say) classifying real car tokens $\{c_1, \dots, c_n\}$ according to their observed attributes ($c_{23} \models_{cars} \text{BMW}$, $c_{41} \models_{cars} \text{Fast}$, ...). The local logics L in this classification \mathcal{C}_{cars} and their associated “car theories” will be fully generated by the data, and therefore if our intuitions about cars are correct the sequent $\text{JapaneseCar}, \text{EuropeanCar} \vdash_L \emptyset$ will be a theorem of every logic L with a non-empty set N of normal car tokens. If we then plugged this classification into the model as just another component (via some appropriate infomorphism) these “empirical” local logics can be lifted up to the system level, and the validity of various candidate abstractions can be checked against them in essentially the same way as we have described higher. The flexibility in which various flavors of logics can be freely combined and interleaved using the “*information flow in distributed systems*” metaphor is one of the original strengths of channel theory, and we believe its potential range of applications as a knowledge representation formalism is not yet fully explored.

The example discussed in this paper was limited to the case of symbolic abstraction (see [78] and [4]). The high degree of generality and modularity of the current approach however leads us to believe that the model can be extended more or less straightforwardly to independently cover arity abstractions and truth abstractions as well, though this will have to be confirmed in subsequent work. In future research we also plan to compare our framework with the theory of abstraction proposed in [4].

References

1. Barwise, J., Seligman, J.: Information Flow. The Logic of Distributed Systems. In: Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, Cambridge (1997)
2. Davis, R., Shrobe, H.E., Szolovits, P.: What is a knowledge representation? AI Magazine 14(1), 17–33 (1993)
3. Ghidini, C., Giunchiglia, F.: Local models semantics, or contextual reasoning = locality + compatibility. Artificial Intelligence 127(2), 221–259 (2001)
4. C. Ghidini and F. Giunchiglia. A semantics for abstraction. In: ECAI proceedings (2004)
5. Giunchiglia, F., Bouquet, P.: Introduction to contextual reasoning. an artificial intelligence perspective. Perspectives on Cognitive Science 3, 138–159 (1997)
6. Giunchiglia, F., Sebastiani, R., Villaforita, A., Walsh, T.: A general purpose reasoner for abstraction. In: Canadian Conference on AI, pp. 323–335 (1996)
7. Giunchiglia, F., Walsh, T.: Using abstraction. In: Proc. of the Eighth Conference of the Society for the Study of Artificial Intelligence and Simulation of Behaviour (AISB’91), pp. 225–234, Leeds, England (1991)
8. Giunchiglia, F., Walsh, T.: A theory of abstraction. Artificial Intelligence 57(2-3), 323–389 (1992)
9. Hobbs, J.R.: Readings in Qualitative Reasoning about Physical Systems. In: Weld, D.S., de Kleer, J. (eds.), pp. 542–545. Kaufmann, San Mateo, CA (1990)
10. Keefe, R.: Vagueness. Cambridge University Press, Cambridge (2000)

11. De Saeger, S., Shimojima, A.: Contextual reasoning in agent systems (Revised Selected and Invited Papers). In: Inoue, K., Satoh, K., Toni, F. (eds.) Computational Logic in Multi-Agent Systems. LNCS (LNAI), vol. 4371, Springer, Heidelberg (2007)
12. Nayak, P.P., Levy, A.: A semantic theory of abstractions. In: Mellish, C. (ed.) Proc. of the Fourteenth International Joint Conference on Artificial Intelligence, pp. 196–203. Morgan Kaufmann, San Francisco (1995)

Approximate Model-Based Diagnosis Using Greedy Stochastic Search

Alexander Feldman¹, Gregory Provan², and Arjan van Gemund¹

¹ Delft University of Technology,
Faculty of Electrical Engineering, Mathematics and Computer Science,
Mekelweg 4, 2628 CD, Delft, The Netherlands
{a.b.feldman,a.j.c.vangemund}@tudelft.nl*

² University College Cork, Department of Computer Science,
College Road, Cork, Ireland
g.provan@cs.ucc.ie**

Abstract. Most algorithms for computing diagnoses within a model-based diagnosis framework are deterministic. Such algorithms guarantee soundness and completeness, but are NP-hard. To overcome this complexity problem, we propose a novel *approximation approach* for multiple-fault diagnosis, based on a greedy stochastic algorithm called SAFARI (StochAstic Fault diagnosis AlgoRithm). SAFARI sacrifices guarantees of optimality, but for models in which component failure modes are defined solely in terms of a deviation from nominal behavior (known as weak fault models), it can compute 80-90% of all cardinality-minimal diagnoses, several orders of magnitude faster than state-of-the-art deterministic algorithms. We have applied this algorithm to the 74XXX and ISCAS-85 suites of benchmark combinatorial circuits, demonstrating order-of-magnitude speedup over a well-known deterministic algorithm, CDA*, for multiple-fault diagnoses.

1 Introduction

Model-Based Diagnosis (MBD) is an area of abductive or model-based inference in which a system model is used, together with observations about system behavior, to isolate sets of faulty components (diagnoses) that explain the observed behavior. The standard MBD formalization [1] frames a diagnostic problem in terms of a set of logical clauses that include mode-variables describing the nominal and fault status of system components; from this the diagnostic status of the system can be computed given an observation (OBS) of the system's sensors. MBD provides a sound and complete approach to enumerating multiple-fault diagnoses, and exact algorithms can guarantee finding a diagnosis optimal with respect to the number of faulty components, probabilistic likelihood, etc.

However, the biggest challenge (and impediment to industrial deployment) is the computational complexity of the MBD problem. The MBD problem of isolating multiple-fault diagnoses is known to be Σ_1^P -complete [2,3]. Since almost

* Supported by STW grant DES.7015.

** Supported by SFI grant 04/IN3/I524.

all proposed MBD algorithms have been complete and exact (with some authors proposing possible trade-offs between completeness and faster consistency checking by employing methods such as BCP [4]), the complexity problem remains a major challenge to MBD.

To overcome this complexity problem, we propose a novel *approximation approach* for multiple-fault diagnosis, based on stochastic algorithms. SAFARI (StochAstic Fault diagnosis AlgoRithm) sacrifices guarantees of optimality, but for diagnostic systems in which faults are described in terms of an arbitrary deviation from nominal behavior SAFARI can compute diagnoses several orders of magnitude faster than competing algorithms.

Our contributions are as follows. (1) This paper introduces an approximation algorithm for computing diagnoses within an MBD framework, based on a greedy stochastic algorithm. (2) We show the theoretical justification for the success of this algorithm, i.e., that minimal-cardinality diagnosis over weak fault models can be solved in poly-time (calling the incomplete SAT-solver BCP), and that more general frameworks are also amenable to this class of algorithm. (3) We apply this algorithm to a suite of benchmark combinatorial circuits, demonstrating order-of-magnitude speedup over a well-known deterministic algorithm, CDA*, for multiple-fault diagnoses. Moreover, whereas the search complexity for the deterministic algorithms tested increases exponentially with fault cardinality, the search complexity for this stochastic algorithm appears to be independent of fault cardinality. SAFARI is of great practical significance, as it can compute a large fraction of cardinality-minimal diagnoses for discrete systems too large or complex to be diagnosed by existing deterministic algorithms.

2 Related Work

MBD is an instance of constraint optimization, with particular constraints over failure variables, as we will describe. MBD has developed algorithms to exploit these domain properties, and our proposed approach differs significantly with almost all MBD algorithms that appear in the literature. While most advanced MBD algorithms make use of preferences, e.g., fault-mode probabilities, to improve search efficiency, the algorithms themselves are deterministic, and use the preferences to identify the most-preferred solutions. This contrasts with stochastic SAT algorithms, which rather than backtracking may randomly flip variable assignments to determine a satisfying assignment.

The most closely-related diagnostic approach is that of Vatan et al. [5], who map the diagnosis problem into the monotone SAT problem, and then propose to use efficient SAT algorithms for computing diagnoses. The approach of Vatan et al. has shown speedups in comparison with other diagnosis algorithms; the main drawback is the number of extra variables and clauses that must be added in the SAT encoding, which is even more significant for strong fault models and multi-valued variables. In contrast, our approach works directly on the given diagnosis model and requires no conversion to another representation.

Stochastic algorithms have been discussed in the framework of constraint satisfaction [6] and Bayesian network inference [7]. The latter two approaches can be used for solving suitably translated MBD problems. It is often the case, though, that these encodings are more difficult for search than specialized ones.

3 Technical Background

The discussion starts by formalizing some basic notions in MBD, extending the notions proposed by de Kleer et al. [8]. A *model* of an artifact is represented as a propositional **Wff** over some set of variables V . Discerning disjoint subsets of them as failure-mode variables (*assumables*) and observable variables (*observables*) gives us a diagnostic system.

Definition 1 (Diagnostic System). A *diagnostic system* DS is defined as the triple $DS = \langle SD, COMPS, OBS \rangle$, where SD is a propositional theory describing the behavior of the system, $COMPS$ is a set of assumable variables in SD , and OBS is a set of some observable variables in SD .

Throughout this paper we will assume that $OBS \cap COMPS = \emptyset$ and $SD \not\models \perp$.

3.1 A Running Example

The Boolean circuit shown in Fig. 1 provides some intuitions behind SAFARI. The subtractor consists of seven components: an inverter, two or-gates, two xor-gates, and two and-gates.

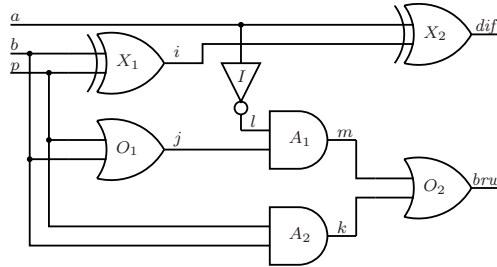


Fig. 1. A subtractor circuit

The expression $h \Rightarrow (o \Leftrightarrow \neg i)$ models an inverter, where the variables i , o , and h represent input, output and health respectively. Similarly, an and-gate is modeled as $h \Rightarrow (o \Leftrightarrow i_1 \wedge i_2)$ and an or-gate is $h \Rightarrow (o \Leftrightarrow i_1 \vee i_2)$. Finally, an xor-gate is specified as $h \Rightarrow (o \Leftrightarrow \neg(i_1 \Leftrightarrow i_2))$.

These propositional formulae are copied for each gate in Fig. 1 and their variables renamed in such a way as to properly connect the circuit and disambiguate the assumables, thus receiving a propositional formula for SD , given by $X_1 \Rightarrow (i \Leftrightarrow \neg(b \Leftrightarrow p))$, $X_2 \Rightarrow (dif \Leftrightarrow \neg(a \Leftrightarrow i))$, $O_1 \Rightarrow (j \Leftrightarrow b \vee p)$, $O_2 \Rightarrow (brw \Leftrightarrow m \vee k)$, $A_1 \Rightarrow (m \Leftrightarrow l \wedge j)$, $A_2 \Rightarrow (k \Leftrightarrow b \wedge p)$, $I \Rightarrow (a \Leftrightarrow \neg l)$. The set of

component (assumable) variables is $\text{COMPS} = \{X_1, X_2, O_1, O_2, A_1, A_2, I\}$. The set of observable variables is $\text{OBS} = \{a, b, p, dif, brw\}$.

3.2 Minimal Diagnosis and Fault Modes

The traditional query in MBD results in finding terms of assumable variables which are explanations for the system description and an observation. The first definition of diagnosis uses a set notation.

Definition 2 (Diagnosis). *Given a system $\text{DS} = \langle \text{SD}, \text{COMPS}, \text{OBS} \rangle$ and an observation term α over the variables in OBS , a diagnosis for DS and α is a set $D \subseteq \text{COMPS}$ such that:*

$$\text{SD} \wedge \alpha \wedge \left[\bigwedge_{c \in D} \neg h_c \right] \wedge \left[\bigwedge_{c \in (\text{COMPS} \setminus D)} h_c \right] \not\models \perp$$

In the MBD literature, a range of types of “preferred” diagnosis have been proposed. The first ordering we consider is a subset-ordering:

Definition 3 (Subset-Minimal Diagnosis). *A diagnosis D is subset-minimal, if no proper subset $D' \subset D$ exists such that D' is also a diagnosis.*

Throughout this paper we interchangeably use a propositional notation for expressing diagnoses. In this case we simply construct a conjunction of literals, each literal having a negative sign if its respective variable is in D and a positive sign otherwise. Consider the example from Fig. [1](#) and an observation $\alpha = a \wedge b \wedge p \wedge \neg dif \wedge \neg brw$. In this case $D_1 = \{A_1, A_2, X_1\}$ is a diagnosis and $D_2 = \{A_1, X_1\}$ is a minimal diagnosis (there are seven more minimal diagnoses for $\text{SD} \wedge \alpha$). Alternatively, instead of D_1 we may write $D'_1 = \neg X_1 \wedge X_2 \wedge O_1 \wedge O_2 \wedge \neg A_1 \wedge \neg A_2 \wedge I$.

The *cardinality* of a diagnosis D is the size of D and is denoted as $|D|$. It represents the number of faulty components in COMPS given SD and α . Next to computing minimal diagnoses, it is of interest to MBD to compute some or all minimal-cardinality diagnoses, given a diagnostic problem.

Definition 4 (Cardinality-Minimal Diagnosis). *A diagnosis D is a cardinality-minimal diagnosis if it is a minimal diagnosis and no other diagnosis D' exists such that $|D| < |D'|$.*

The cardinality of a cardinality-minimal diagnosis computed from a system description SD and an observation α is denoted as $\text{MinCard}(\text{SD} \wedge \alpha)$. For our example and the observation $\alpha = a \wedge b \wedge p \wedge \neg dif \wedge \neg brw$, it follows that $\text{MinCard}(\text{SD} \wedge \alpha) = 2$. Note that in this case all minimal diagnoses are also cardinality-minimal diagnoses.

There are subset-minimal diagnoses which are not cardinality-minimal diagnoses. Consider, for example, the diagnostic system $\text{DS} = \langle \text{SD}, \text{COMPS}, \text{OBS} \rangle$, where $\text{SD} = (h_1 \wedge h_2 \wedge x) \vee (h_4 \wedge x)$, $\text{COMPS} = \{h_1, h_2, h_3, h_4\}$, $\text{OBS} = \{x\}$, and $\alpha = x$. In this case, $D_1 = \{h_1, h_2, h_3\}$ is a non-subset-minimal diagnosis,

$D_2 = \{h_1, h_2\}$ and $D_3 = \{h_4\}$ are subset-minimal diagnoses, but only D_3 is a cardinality-minimal diagnosis.

Another important diagnosis preference relation that we consider is one induced by a probability distribution over failure mode instantiations, γ , i.e., $Pr : \gamma \rightarrow [0, 1]$. If we assume that all components fail independently, then the probability of a multiple-fault F is just the product of the component probabilities, i.e., $Pr(F) = \prod_{F_i \in F} Pr(F_i)$.

Definition 5 (Probability-Minimal Diagnosis). *Given a non-trivial probability assignment to component failure modes, a probability-maximal diagnosis ω is a fault-mode such that \nexists any other diagnosis ω' such that $Pr(\omega') > Pr(\omega)$.*

In the following, we will focus on subset-minimal and cardinality-minimal diagnoses; these two relationships mean that our results will also hold for a probability-minimal diagnosis.

MBD defines two broad classes of fault models, based on weak and strong modeling assumptions for abnormal behavior.

Weak-fault models define normative behavior of their components only, i.e., models which specify no fault-modes.

Definition 6 (Weak Fault Model). *A diagnostic system DS belongs to the class **WFM** iff SD is in the form $(h_1 \Rightarrow F_1) \wedge \dots \wedge (h_n \Rightarrow F_n)$ such that for $1 \leq i, j \leq n$, $\{h_i\} \subseteq \text{COMPS}$, $F_j \in \mathbf{Wff}$, and none of h_i appears in F_j .*

In contrast, *strong fault models* specify the faulty behavior of their components. One way to define such a model is by partitioning the set of observable variables OBS into two subsets IN and OUT (denoting input and output variables respectively). Defining values to IN and COMPS then allows us to find a *unique assignment* to the values in OUT.

Definition 7 (Strong Fault Model). *Given a system DS and a partition $\text{OBS} = \text{IN} \cup \text{OUT}$, $\text{DS} \in \mathbf{SFM}$ if for any instantiation ϕ of all variables in $\text{IN} \cup \text{COMPS}$, it holds that there is exactly one term ψ such that $\phi \models \text{SD} \wedge \psi$ and ψ is an instantiation of all variables in OUT.*

In the following we show how our stochastic algorithm can compute subset- and cardinality-minimal diagnoses for SD such that $\text{SD} \in \mathbf{WFM}$, and indicate how the algorithm can be generalized to cover $\text{SD} \in \mathbf{SFM}$.

4 Stochastic MBD Algorithm

In this section we discuss an algorithm for computing multiple-fault diagnoses using stochastic search.

4.1 A Simple Example (Continued)

We now show what happens when we apply A^* and stochastic search to our running example.

A deterministic A* search for the above diagnosis discovers it after expanding 127 nodes and performing 19 consistency checks. Even enabling conflict focusing may not result in a small number of generated nodes and consistency checks (this depends on the model, observation and conflict extraction mechanism), which shows how deterministic diagnosis search becomes impractical with bigger systems.

We will now show a two-step diagnostic process that requires fewer variable assignments and consistency checks. Step 1 involves randomly choosing candidates. Step 2 attempts to minimize the fault cardinality in these candidates.

In step 1, the stochastic diagnostic search for the subtractor example will start from a random quintuple candidate¹. In this particular version of our algorithm, once a component is marked as healthy, it cannot be changed back to faulty. To compensate for that, we perform multiple restarts from a random candidate. In our subtractor example and for $\alpha = a \wedge b \wedge p \wedge \neg dif \wedge \neg brw$, if $X_1 \wedge X_2$ is in the initial “guess” it will prove inconsistent with $SD \wedge \alpha$ and another quintuple fault candidate will be guessed.

Assume that the second candidate is $\omega'_2 = \neg X_1 \wedge \neg X_2 \wedge O_1 \wedge \neg O_2 \wedge \neg A_1 \wedge \neg A_2 \wedge I$. Clearly, $SD \wedge \alpha \wedge \omega' \not\models \perp$. The search algorithm may next try to improve the diagnosis by “flipping” A_2 . The candidate $\omega'_3 = \neg X_1 \wedge \neg X_2 \wedge O_1 \wedge \neg O_2 \wedge \neg A_1 \wedge A_2 \wedge I$ is a valid quadruple fault diagnosis and it can be improved twice more by “flipping” X_2 and O_2 . This gives us the final double-fault $\omega'_6 = \neg X_1 \wedge X_2 \wedge O_1 \wedge O_2 \wedge \neg A_1 \wedge A_2 \wedge I$. The actual algorithm is somewhat more involved as during the variable flipping it is normal to find inconsistencies. Instead of restarting, it will simply discard these inconsistent candidates until some termination criterion is satisfied.

Intuitively, from our example, due to the large number of double fault diagnoses explaining the same observation, it is not difficult to randomly guess sequences of variables which need to be false in order to explain the observation.

4.2 A Greedy Stochastic Algorithm

The greedy stochastic algorithm, which we introduce next, finds multiple cardinality-minimal diagnoses (if such exist).

The stochastic algorithm presented in this paper uses the valuation function $Pr : \gamma \rightarrow [0, 1]$. In the analysis of our algorithm we use Pr to determine if an assignment to a health variable denotes a failure or a healthy mode. The complexity of the algorithm presented in this paper is not sensitive to Pr and the only use of the probabilities is to guide the search for a better efficiency.

The randomized search process performed by SAFARI has two parameters, M and N . There are N independent searches that start from randomly generated candidates. After an initial candidate ω is found to be consistent with $SD \wedge \alpha$, i.e., ω is a diagnosis, the algorithm tries to improve the cardinality of the diagnosis (while preserving its consistency) by randomly “flipping” fault literals.

¹ In the formal description of the algorithm we describe a method for determining the initial candidates.

Algorithm 1. SAFARI: A stochastic hill climbing algorithm for approximating a set of cardinality-minimal diagnoses.

```

1: function HILLCLIMB(DS,  $\alpha$ ,  $M$ ,  $N$ ,  $Pr$ ) returns a trie
   inputs: DS =  $\langle$ SD, COMPS, OBS $\rangle$ , a diagnostic system
            $\alpha$ , term, observation
            $M$ , integer, climb restart limit
            $N$ , integer, number of tries
            $Pr$ , a valuation function
   local variables:  $m, n$ , integers
                    $\omega, \omega'$ , terms
                    $R$ , a trie
2:    $n \leftarrow 0$ 
3:   while  $n < N$  do
4:      $\omega \leftarrow \text{RANDOMCANDIDATE}(Pr)$ 
5:     if  $SD \wedge \alpha \wedge \omega \not\models \perp$  then
6:        $m \leftarrow 0$ 
7:       while  $m < M$  do
8:          $\omega' \leftarrow \text{IMPROVEDIAGNOSIS}(Pr, \omega)$ 
9:         if  $SD \wedge \alpha \wedge \omega' \not\models \perp$  then
10:           $\omega \leftarrow \omega'$ 
11:           $m \leftarrow 0$ 
12:        else
13:           $m \leftarrow m + 1$ 
14:        end if
15:      end while
16:      unless  $\text{ISSUBSUMED}(R, \omega)$  then
17:         $\text{ADDTOTRIE}(R, \omega)$ 
18:         $\text{REMOVESUBSUMED}(R, \omega)$ 
19:      end unless
20:       $n \leftarrow n + 1$ 
21:    end if
22:  end while
23:  return  $R$ 
24: end function

```

Each attempt to find a cardinality-minimal diagnosis terminates after M unsuccessful attempts to change the value of a fault variable to healthy state. Thus, increasing M will lead to a better exploitation of the search space and possibly diagnoses of lower cardinality, while decreasing it will improve the overall speed of the algorithm.

Similar to deterministic methods for MBD, SAFARI uses a SAT-based procedure for checking the consistency of $SD \wedge \alpha \wedge \omega$. Because SD and α do not change in consistency checks, using an LTMS [9] can improve search efficiency. The implementation of SAFARI combines a BCP-based LTMS to check for inconsistencies. If a candidate is consistent, a second DPLL-based check is invoked for completeness.

The `RANDOMCANDIDATE` function generates a candidate diagnosis, used for “seeding” the diagnostic search. The valuation function can be modified to provide a more informed starting point, thus decreasing the number of “climbing” steps. The initial diagnosis ω should be of high cardinality, to increase the likelihood of $\text{SD} \wedge \alpha \wedge \omega \not\models \perp$. In order to do that, we generate an instantiation of ω by using Pr and *scaling* the a priori probabilities in Pr to bias the probability density function (pdf) from which we draw the initial candidates. Consider an example in which each component $h \in \text{COMPS}$ fails with a probability of 5%. The valuation function is $Pr(h = \mathbf{False}) = 0.05$. We may use a scaling coefficient $k = 5$ which would lead to `RANDOMCANDIDATE` returning a candidate with a quarter of the components failing.

The biasing of Pr can improve the efficiency of `SAFARI` by exploiting knowledge about the likelihood of the cardinality of the cardinality-minimal diagnosis. In particular, when expecting cardinality-minimal diagnoses of high cardinality Pr should be configured to return an initial fault of higher cardinality. If the expected faults are of small cardinality, the search may start from a candidate with fewer faulty components, in which case more attempts (increased N) would be necessary to find local diagnoses close to the global optimum.

The `IMPROVEDIAGNOSIS` function generates a candidate ω' of smaller cardinality than the diagnosis ω , supplied as an argument. This is done by flipping a random faulty literal in ω . The probability of flipping a faulty literal l in ω is inverse proportional to the a priori probability $Pr(l)$. Consider a diagnosis $\omega = \neg h_1 \wedge \neg h_2 \wedge \neg h_3 \wedge \neg h_4$, where $Pr(h_1 = \mathbf{False}) = Pr(h_2 = \mathbf{False}) = 0.1$ and $Pr(h_3 = \mathbf{False}) = Pr(h_4 = \mathbf{False}) = 0.025$. In this case `IMPROVEDIAGNOSIS` would return $\omega' = h_1 \wedge \neg h_2 \wedge \neg h_3 \wedge \neg h_4$ or $\omega' = \neg h_1 \wedge h_2 \wedge \neg h_3 \wedge \neg h_4$, each of the two with probability of 0.4, and $\omega' = \neg h_1 \wedge \neg h_2 \wedge h_3 \wedge \neg h_4$ or $\omega' = \neg h_1 \wedge \neg h_2 \wedge \neg h_3 \wedge h_4$ the latter with probability 0.1.

There is no guarantee that two diagnostic searches, starting from a random diagnoses, would not lead to the same cardinality-minimal diagnosis. To prevent this, we store the generated diagnoses in a trie R [10], from which it is straightforward to extract the resulting diagnoses by recursively visiting its nodes. A diagnosis ω is added to the trie R by the function `ADDTOTRIE`, iff no subsuming diagnosis is contained in R (the `ISSUBSUMED` subroutine checks on that condition). After adding a diagnosis ω to the resulting trie R , all diagnoses contained in R and subsumed by ω are removed by a call to `REMOVESUBSUMED`.

5 Optimality and Complexity of Greedy Stochastic Algorithm

One of the key factors in the success of the proposed algorithm is the exploitation of the continuity of the search-space of weak fault models, where by continuity we mean that we can monotonically reduce the cardinality a non-minimal diagnosis. This section shows that our algorithm can guarantee finding minimal diagnoses in weak fault models in polynomial time (given a SAT oracle such as `BCP`), and

trades off optimality in more general diagnostic frameworks, such as cardinality-minimal diagnostic inference or strong fault models.

5.1 Cardinality-Minimal Diagnosis in Weak-Fault Models

We will show some properties of minimal diagnoses. These properties are also true for fault-modes with a slight adaptation of the notation.

Hypothesis 1 (Minimal Diagnosis Hypothesis) *Let SD be a system description and D' a diagnosis. The Minimal Diagnosis Hypothesis (MDH) holds in SD if for any D such that $D \supset D'$ it holds that D is also a diagnosis.*

It has been shown in [8] that if a model $SD \in \mathbf{WFM}$ (cf. Def. [6]), then the Minimal Diagnosis Hypothesis (MDH) holds. There are other theories $SD' \notin \mathbf{WFM}$ for which MDH holds. Unfortunately, no necessary condition is known for MDH to hold in an arbitrary SD' .

Using MDH together with Def. [3], if $SD \in \mathbf{WFM}$ then we immediately have the following lemma.

Lemma 1. *If D is a diagnosis of a diagnosis problem DS and MDH holds for DS , then there is a subset-minimal diagnosis D' that subsumes D , i.e., $D' \subseteq D$.*

Our greedy algorithm starts with a “seed” diagnosis and then randomly flips faulty component variables. We now use the MDH property to show that, starting with a non-minimal diagnosis D , the greedy stochastic diagnosis algorithm can monotonically reduce the size of “seed” diagnosis to obtain a minimal diagnosis through appropriately flipping a fault variable from faulty to healthy; if we view this flipping as search, then this search is continuous in the diagnosis space [2].

Theorem 1. *Given a weak fault model $SD \in \mathbf{WFM}$ and a non-subset-minimal diagnosis D with $|D| = \mu \leq n$ faulty components, the greedy stochastic diagnosis algorithm is guaranteed to compute a minimal diagnosis.*

We can now prove the following correctness result:

Theorem 2. *The greedy stochastic diagnosis algorithm is guaranteed to compute a subset-minimal diagnosis for a weak fault model with $|\text{OBS}| = n$ in $O(\Theta n)$ time, where $O(\Theta)$ is the complexity of a consistency check.*

Since consistency-checking for this model class can be done in polynomial time (using BCP propagation), we have just demonstrated a polynomial-time algorithm for computing minimal diagnoses in a weak-fault-model system description (\mathbf{WFM}). This result has been shown in the literature for a divide-and-conquer approach [12], but to our knowledge no current algorithm takes advantage of this approach.

We now provide a brief intuition behind the performance of the SAFARI algorithm. A formal derivation of the performance/cost model is beyond the scope of this paper and appears in [11].

² We omit proofs due to space limitations, but refer the reader to [11] for all proofs.

Assume we have an MBD problem with S optimal diagnoses of cardinality C . A key to the good performance of the algorithm is the fact that the pdf of the number of successful consecutive steps $j = 0, 1, \dots$ toward the optimal solution ($j = |\text{COMPS}| - C$), is generally an *increasing* function, rather than decreasing; i.e., the probability mass is concentrated near the optimum solution. This somewhat counter-intuitive fact is caused by two phenomena. First, the retry mechanism has a huge impact on the shape (slope) of the pdf. Whereas for $M = 0$ (no retries) the pdf is still close to a geometric distribution (i.e., a *decreasing* function), even for relatively small M (e.g., $M = 4$, as in our experiments) the retry process turns the slope into an increasing function. Second, the fact that there are usually multiple ($S > 1$) C -fault diagnoses implies that the attained j value is essentially the *maximum* of the j values per individual C -fault solution. The associated, order-statistical effect further increases the positive slope of the pdf. Thus, even for one run, the probability of reaching the optimum solution is quite high.

The effect of N (taking the maximum of the attained j over multiple runs) also translates into an order-statistical effect. However, this third effect is not nearly as powerful as the effect of M (and S). Hence, optimizing SAFARI's performance/cost ratio is primarily a matter of increasing M , rather than N .

Finally, note that the number of retries also guarantees optimality in those cases where the attained solution cardinality is less or equal to M . For instance, when a solution with $C = 4$ is found (where $M = 4$), it must be optimal, as the remaining of lower-cardinality solutions have all been (unsuccessfully) tried.

5.2 More General Diagnostic Frameworks

This section now addresses how the algorithm will perform in more general diagnostic frameworks, such as computing cardinality-minimal diagnoses or dealing with strong fault models. In generalizing beyond subset-minimal diagnosis computation, less restrictive definitions are computationally harder [3].

We can modify SAFARI to compute a wide variety of diagnoses, e.g., subset-minimal, non-minimal, cardinality-minimal, etc., by modifying the type of diagnosis computed together with the trie maintenance and subsumption testing of lines 16-18 of the pseudo-code.

The complexity of SAFARI is derived in a straightforward way.

Proposition 1. *The time complexity of Alg. 1 is $O(MN \log N\Theta)$, where Θ is the complexity of the consistency checking procedure.*

The $\log N$ factor comes from the trie maintenance, which contains a maximum number of N diagnoses with some ordering imposed on their literals. Note that the average case complexity of consistency checking, although exponential in the worst case, is low polynomial when incomplete methods like BCP are used [13] or when the model is highly-observable.

In this more general case we have no completeness guarantee, as we do for subset-minimal diagnoses. Note that this is effectively a polynomial-time algorithm that trades off some small amount of completeness and optimality for sig-

nificant improvements in efficiency relative to deterministic diagnosis algorithms. In these more general frameworks, from a theoretical perspective one can only provide probabilistic arguments about the likelihood of finding particular classes of diagnosis; this is a topic of future work. The experimental results presented later in this article show that significant speedups over complete algorithms are possible while losing relatively little diagnostic completeness.

In generalizing from weak to strong fault models, the key difference is the increased difficulty in “guessing” the initial diagnosis for SAFARI. For a weak fault model, we are guaranteed to find a subset-minimal diagnosis by choosing an initial diagnosis with all components faulty³ but this guess is not guaranteed to be consistent in a strong fault model. Developing a robust diagnosis initialization algorithm for strong fault models is a topic for future research.

6 Experimental Results

Next, we discuss some empirical results measured from an implementation of SAFARI. In the following, for any models SD, it holds that $SD \in \mathbf{WFM}$.

We have implemented SAFARI in approximately 700 lines of C code (excluding LTMS and DPLL) and it is a part of the LYDIA⁴ package.

Table 1. Test model sizes

Name	Description	H	V	C_w	O
74182	4-bit CLA	19	47	75	14
74283	4-bit adder	40	89	130	14
74L85	4-bit comparator	41	93	134	14
74181	4-bit ALU	62	138	216	22
c432	27-channel interrupt controller	160	356	514	43
c499	32-bit SEC circuit	202	445	714	73
c880	8-bit ALU	383	826	1 112	86
c1355	32-bit SEC circuit	546	1 133	1 610	73
c1908	16-bit SEC/DEC	880	1 793	2 378	58
c2670	12-bit ALU	1 193	2 543	3 269	221
c3540	8-bit ALU	1 669	3 388	4 608	72
c5315	9-bit ALU	2 307	4 792	6 693	301
c6288	32-bit multiplier	2 416	4 864	7 216	64
c7552	32-bit adder	3 512	7 230	9 656	313

Table 1 summarizes the benchmark suite. All models are derived from the 74XXX and ISCAS85 family of benchmark circuits. We have added an assumable variable to each gate in each model. The benchmark implements weak fault models for

³ We start with half the components faulty and use the parameter M to restart if our initial guess is incorrect.

⁴ LYDIA can be downloaded from <http://fdir.org/lydia/>

each component, in a way similar to the example. The same valuation function Pr has been used in all the experiments. In particular, $Pr(h = \mathbf{False}) = 0.01$, and $Pr(h = \mathbf{True}) = 0.99$.

The number of assumable variables is denoted as H . The total number of variables is denoted V and the number of clauses in the CNF representation is denoted as C_w . The number of observable variables is denoted as O .

All the experiments described in this paper are performed on a host with 1.86 GHz Pentium M CPU and 2 Gb of RAM.

6.1 Comparison to HA* and Multiple-Fault Scalability

We compare the efficiency of SAFARI to HA* [14] with Max-Fault Min-Cardinality (MFMC) observation vectors [15]. MFMC observation vectors maximize the number of faults in the cardinality-minimal diagnoses consistent with a model. From the deterministic algorithms, HA* performs better than CDA* [4] in finding cardinality-minimal diagnoses of high cardinality. On the other hand, CDA* is very fast in finding faults of small cardinality (single and double faults) and we will then compare SAFARI to CDA*.

First, we use small models with observations maximizing the number of faults in a cardinality-minimal diagnosis. The results, shown in Table 2, illustrate an advantage of SAFARI: its performance does not degrade when the fault cardinality increases. We have run two groups of experiments: finding a single cardinality-minimal diagnosis and finding all cardinality-minimal diagnoses.

Table 2. Times [ms] for diagnosing MFMC faults by Alg. 1 and HDA*

Name	Single Diagnosis					Multiple Diagnoses				
	MFMC	T_h	T	C	K	T'_h	T'	K'	N'	M'
74182	5	38	2	5	300	171	143	242	800	4
74283	5	4708	4	5.5	814	27606	5759	665	10000	4
74L85	3	143	4	3	100	1281	184	90	400	4
74181	7	106386	9	7	3817	634739	31377	3236.7	20000	8

Next, we describe the notation in the column headings of Table 2. MFMC is the number of faults in the cardinality-minimal diagnosis consistent with the MFMC observation. T_h is the time for finding a single diagnosis by the HA* algorithm. T is the time for finding a single diagnosis by Alg. 1. C is the cardinality of the diagnosis generated by SAFARI. K is the number of cardinality-minimal diagnoses as counted by the deterministic algorithm HA*. The time for finding all of these diagnoses by HA* is denoted as T'_h . T' is the time for SAFARI to find K' multiple cardinality-minimal diagnoses.

SAFARI is designed to find multiple cardinality-minimal diagnosis. In these experiments, we have configured it with a number N of runs in order to return a small number of diagnoses, and we have ignored all but the first diagnosis.

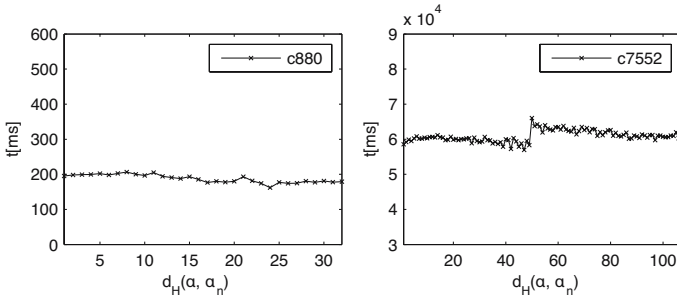


Fig. 2. Diagnosis time of SAFARI with multiple observation vectors

We performed the single fault experiments, shown in Table 2, with $N = 8$ and $M = 4$. For the multiple diagnoses, the algorithm is configured with $M = M'$ and $N = N'$ as described in Table 2.

We have averaged the results of all the experiments involving SAFARI over 10 runs. SAFARI is a local search algorithm, and hence it can compute a suboptimal diagnosis. This was the case in the 74283 model experiments, in which 5 out of 10 runs returned a cardinality-minimal diagnosis with 6 faults, while the global optimum has 5 faults, resulting in the 5.5 value for C in Table 2.

Table 2 demonstrates the main advantage of SAFARI, that its performance does not depend on the number of faults in the cardinality-minimal diagnoses. Furthermore, SAFARI finds a good coverage of all cardinality-minimal diagnoses. This coverage varies from 81% in 74182 to 90% in 74L85.

We have also run SAFARI on bigger circuits. Figure 2 shows the time for finding multiple-fault diagnosis for c880 and c7552. The diagnosis was run k times where k is the number of outputs in each circuit. For run $x = 0$, we have assigned random values to the inputs and computed (by using propagation) the values of all the outputs. For $x = 1$ we have flipped one output in α , for $x = 2$ two outputs, etc. Thus on the horizontal axis in Fig. 2 we have the Hamming distance between α and an observation consistent with a no-fault (nominal) diagnosis. Again, SAFARI showed no dependency on α , only a small increase in the diagnostic time for c7552 due to the difficulty of finding an initial diagnosis. The latter can be easily overcome by scaling the initial a-priori probabilities.

6.2 Comparison to CDA*

Table 3 shows the result from finding single and double faults with arbitrary (manually computed) observations. Finding single faults is known to be trivial in MBD and CDA* performs well on these simple problems. The time for finding a single fault by CDA* is shown in column T_1^* . The time for the CDA* algorithm to find a double fault is shown in column T_2^* . The CDA* algorithm could not compute a double fault diagnosis in less than 10 min time for the five biggest circuits, in which cases we have interrupted the search.

Table 3. Running times [ms] of CDA* and Alg. [□](#)

Name	Single-Fault			Double-Fault		
	T_1^*	T_1	C_1	T_2^*	T_2	C_2
c432	9	32	1	5	34	2
c499	3	53	1	152	64	2
c1908	34	95	1	509	94	2
c880	18	186	1	62 068	186	2
c1355	11	285	1	4 300	310	2
c2670	1 425	1 362	1	–	1 352	2
c3540	3 050	3 080	1	–	3 115	2
c5315	13 849	19 322	1	–	19 764	2
c6288	18 317	11 070	1.4	–	11 366	2.2
c7552	35 801	37 269	1	–	37 585	2.2

The times for the stochastic algorithm to discover a single and a double fault are denoted as T_1 and T_2 respectively. We have used $M = 4$ and $N = 8$ for the search, that is, maximum number of four retries before giving up the climb, and a total of 4 attempts. In some of the cases, our stochastic algorithm could not find a cardinality-minimal diagnosis, but a suboptimal one. We have shown the cardinality of the results for single and double faults in columns C_1 and C_2 , respectively. Again, the values of T_1 , C_1 , T_2 , and C_2 are averaged over 10 runs.

The relatively small number of restarts lead to small overall search time and in a very few cases to suboptimal result for the diagnosis cardinality. Increasing N would lead to finding a global cardinality-minimal diagnosis in all the cases. We note that increasing M would not help to finding a diagnosis of lower cardinality.

As is visible from Table [3](#), in the single fault scenario, CDA* performs better than the stochastic algorithm, which is not surprising as in CDA* all single fault candidates are tested first. On the other hand, the stochastic method performed 8 independent attempts to find a cardinality-minimal diagnosis which, having the overhead of consistency checking, led to the slightly worse performance for computing single fault diagnoses.

Again, the performance of SAFARI does not degrade when the number of faults increases. This is not the case with deterministic algorithms like CDA* or HA*. The time for the stochastic algorithm to find a double fault is the same as for finding a single fault, while CDA* suffers from a combinatorial explosion.

7 Conclusion and Future Work

We have described a greedy stochastic algorithm for computing diagnoses within a model-based diagnosis framework. We have shown that subset-minimal diagnoses can be computed optimally in weak fault models, and that almost all cardinality-minimal diagnoses can be computed for more general fault models.

We have applied this algorithm to a suite of benchmark combinatorial circuits encoded using weak fault models, and shown significant performance improvements for multiple-fault diagnoses, compared to a well-known deterministic algorithm, CDA*. Our results indicate that, although the greedy stochastic algorithm is outperformed for the single-fault diagnoses, it shows at least an order-of-magnitude speedup over CDA* for multiple-fault diagnoses. Moreover, whereas the search complexity for the deterministic algorithms tested increases exponentially with fault cardinality, the search complexity for this stochastic algorithm appears to be independent of fault cardinality.

We have demonstrated the superior performance (over deterministic algorithms) of SAFARI for the class of discrete circuits specified using weak fault models. We argue that SAFARI can be of broad practical significance, as it can compute a significant fraction of cardinality-minimal diagnoses for systems too large or complex to be diagnosed by existing deterministic algorithms.

In future work, we plan to experiment on models with a combination of weak and strong failure-mode descriptions. We also plan on experimenting with a wider variety of stochastic methods, such as simulated annealing and genetic search, using a larger set of benchmark models. Last, we plan to apply our algorithms to a wider class of abduction and constraint optimization problems.

References

1. Reiter, R.: A theory of diagnosis from first principles. *Artificial Intelligence* 32(1), 57–95 (1987)
2. Bylander, T., Allemang, D., Tanner, M., Josephson, J.: The computational complexity of abduction. *Artificial Intelligence* 49, 25–60 (1991)
3. Friedrich, G., Gottlob, G., Nejd, W.: Physical impossibility instead of fault models. In: *Proc. AAAI* (1990)
4. Williams, B., Ragno, R.: Conflict-directed A* and its role in model-based embedded systems. *Journal of Discrete Applied Mathematics* (2004)
5. Vatan, F., Barrett, A., James, M., Williams, C., Mackey, R.: A novel model-based diagnosis engine: Theory and applications. In: *IEEE Aerospace Conf., IEEE Computer Society Press, Los Alamitos* (2003)
6. Freuder, E.C., Dechter, R., Ginsberg, B., Selman, B., Tsang, E.P.K.: Systematic versus stochastic constraint satisfaction. In: *Proc. IJCAI 95, vol. 2* (1995)
7. Kask, K., Dechter, R.: Stochastic local search for Bayesian networks. In: *Proc. AISTAT'99* (1999)
8. de Kleer, J., Mackworth, A., Reiter, R.: Characterizing diagnoses and systems. *Artificial Intelligence* 56(8), 197–222 (1992)
9. McAllester, D.: Truth maintenance. In: *Proc. AAAI'90, vol. 2* (1990)
10. Forbus, K., de Kleer, J.: *Building Problem Solvers*. MIT Press, Cambridge (1993)
11. Feldman, A., Provan, G., van Gemund, A.: On the performance of Safari algorithms. Technical Report TUD-SERG-2007-011, TU Delft (2007)
12. Mozetič, I.: A polynomial-time algorithm for model-based diagnosis. In: *Proc. ECAI'92, pp. 729–733* (1992)

13. Zabih, R., McAllester, D.: A rearrangement search strategy for determining propositional satisfiability. In: Proc. AAAI'88, pp. 155–160 (1988)
14. Feldman, A., van Gemund, A.: A two-step hierarchical algorithm for model-based diagnosis. In: Proc. AAAI'06 (July 2006)
15. Feldman, A., Provan, G., van Gemund, A.: Generating manifestations of max-fault min-cardinality diagnoses. In: Proc. DX'07 (May 2007)

Combining Perimeter Search and Pattern Database Abstractions

Ariel Felner and Nir Ofek

Deutsche Telekom AG labs at Ben-Gurion University of the Negev, Beer-Sheva, 84105
{felner, nirofek}@bgu.ac.il

Abstract. A *pattern database* abstraction (PDB) is a heuristic function in a form of a lookup table. A PDB stores the cost of optimal solutions for instances of abstract problems (subproblems). These costs are used as admissible heuristics for the original problem. Perimeter search (PS) is a form of bidirectional search. First, a breadth-first search is performed backwards from the goal state. Then, a forward search is executed towards the nodes of the perimeter. In this paper we study the effect of combining these two techniques. We describe two methods for doing this. The simplified method uses a regular PDB (towards a single goal state) but uses the perimeter to correct heuristics of nodes outside the perimeter. The second, more advanced method is to build a PDB that stores the cost of reaching any node of the perimeter from a given pattern. Although one might see great potential for speedup in the advanced method, we theoretically show that surprisingly most of the benefit of combining perimeter and PDBs is already exploited by the first method. We also provide experimental results that confirm our findings. We then study the behavior of our new approach when combined with methods for using multiple PDBs such as maxing and adding.

1 Introduction and Overview

Heuristic search algorithms such as A* and IDA* find optimal solutions to state-space search problems. They are guided by the cost function $f(n) = g(n) + h(n)$, where $g(n)$ is the actual distance from the initial state to state n and $h(n)$ is a heuristic function estimating the cost from n to a goal state. If $h(s)$ is “admissible” (i.e., is always a lower bound) then these algorithms are guaranteed to find optimal paths.

Two general directions are usually taken in order to reduce the search effort. The first direction is to introduce advanced search algorithms that better decide which node of the search tree to expand next. The second direction is the developing of more accurate heuristic functions. Hopefully, the new heuristics will produce larger f -values for many nodes of the search tree and more subtrees in the search tree will be pruned.

An interesting work in the first direction is *Perimeter Search* (PS) where the search is performed towards a set of nodes surrounding the goal state [3,12]. This set of nodes (called the *perimeter*) is generated in a preprocessing phase by performing a breadth-first search backwards from the goal state up to a given depth d . In addition, a new technique for obtaining accurate heuristic function is by using *pattern database* abstractions (PDBs) [2]. PDBs are lookup tables that store the optimal solutions for all instances of a simplified problem abstracted from the original problem. These values are used as admissible heuristics for the original (non abstracted) problem.

The two directions of perimeter search and PDBs are orthogonal. In this paper we study the effect of combining these two techniques and introduce two methods for doing this. In the first, simplified method we generate a PDB in the regular way and use the perimeter nodes to correct mistaken low heuristic values. In the second method, instead of building a regular PDB with a single goal state, we use the perimeter nodes as a set of multiple goals and build a PDB to predict the distance to the closest node on the perimeter. At a first glance, one might see a great potential for further reduction in the search effort when using the second method. Indeed, during the past years, many researchers informally suggested that this idea should be explored as the potential for speedup in the search seems large¹. Surprisingly, we found that there are only limited or no benefits in the second method and that most of the potential benefit of combining perimeter and PDBs is already exploited by the first method. In this paper we theoretically explain this phenomenon and provide experimental results to confirm it. We then study the behavior of our new approach when combined with methods for using multiple PDBs such as maxing and adding.

2 Background

We begin with providing some background on the different techniques that are studied in this paper. We first describe the TopSpin problem which is our leading testbed in this paper.

2.1 The TopSpin Domain

The (N, K) -TopSpin puzzle has N tokens arranged in a ring. Any set of K consecutive tokens can be reversed (rotated 180 degrees in the physical puzzle). Our encoding of this puzzle has N operators, one for each possible reversal/rotation. Each operator has a cost of one. The $(14, 4)$ -TopSpin puzzle is shown in Figure 1 in its goal state. The bold frame indicates a possible reversal of 4 adjacent tokens. There are $n!$ different possible ways to permute the tokens into the locations. However, since the puzzle is cyclic only the relative location of the different tokens matters and thus there are only $(n - 1)!$ different states in practice.

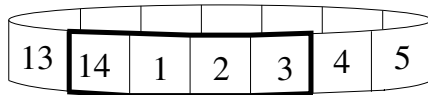


Fig. 1. $(9,4)$ -TopSpin states

2.2 Problem Spaces

A *problem space* is usually described abstractly as a set of atomic states, and a set of operators that map states to states. In addition, a specific problem instance is a problem

¹ Personal discussion with a number of different researchers.

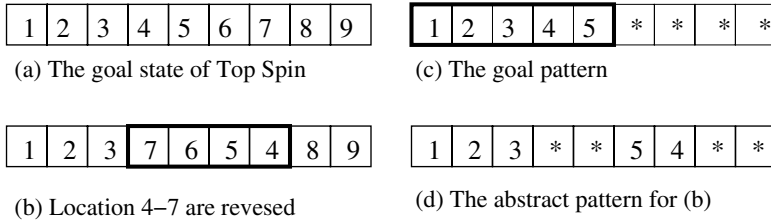


Fig. 2. (9,4)-TopSpin states

space together with a particular initial state and a (set of) goal state(s). The task is to find an optimal path from the initial state to a goal state.

A *state* in problem spaces can be usually described (for *combinatorial problems*) as a vector of *state variables*, each of which is assigned a particular *value*. For the Top-Spin puzzle, there is a variable for each token, whose value indicates its position.

An *operator* in this formulation is a partial function from a state vector to a state vector. An operator changes the values of some of the variables, 4 in the case of TopSpin used in this paper.

The *goal state* is a specified state or a set of states. For problems, such as Top-Spin, the goal state is usually a canonical state where object i is in location i . This goal state is shown in figure 2a.

2.3 Pattern Database Abstractions

Given a vector of state variables, a subset of the variables defines an *abstract problem* (or *subproblem*) where we only assign values to variables in the subset, called *pattern variables*, while the values of the other remaining variables are treated as *don't cares*. For example, in Top-Spin, a subproblem would only include a subset of the tokens (e.g., 1-5 as in figure 2) and ignore the others.

A *pattern* (abstract state) is a specific assignment of values to the pattern variables. The *pattern space* or *abstract space* is the set of all the different reachable patterns of a given abstract problem.

Each state in the original state space is *projected/abstracted* to a pattern of the pattern space by only considering the pattern variables, and ignoring the others. The *goal pattern* is the pattern which is the projection of the goal state. Figure 2 shows states and their projected patterns for the (14, 4)-TopSpin.

There is an *edge* between two different patterns p_1 and p_2 in the pattern space if and only if there exist two states s_1 and s_2 of the original problem, such that p_1 is the projection of s_1 , p_2 is the projection of s_2 , and there is an operator of the original problem space that connects s_1 to s_2 .

The shortest distance between two patterns p_1 and p_2 in the pattern space is therefore a lower bound on the shortest distance in the original space between any pair of states s_1 and s_2 such that p_1 is the projection of s_1 and p_2 is the projection of s_2 .

A *pattern database* (PDB) is a lookup table that includes an entry for each pattern of the pattern space. The value stored for a pattern p is the distance in the pattern space

from p to the goal pattern. A PDB value stored for a given pattern is therefore an admissible heuristic for all the states that are projected to that pattern. PDBs were first introduced by Culberson and Schaeffer in the context of the 15 puzzle [2].

Typically, a PDB is built in a preprocessing phase by searching backwards, breadth-first, from the goal pattern until the entire pattern space is spanned. Given a state S in the original space, an admissible heuristic value for S , $h(S)$, is computed using a pattern database in two steps. First, S is projected to the corresponding pattern. Then, this pattern is looked up in the PDB and the corresponding value is returned as the value for $h(S)$.

PDBs have proven very useful in optimally solving combinatorial puzzles and other problems [2,9,10,6,8,4,5,14,13]. In many cases a number of different PDBs can be built. During the search, we might consult them all and take their maximum as the heuristic [9,7]. Furthermore, in special circumstances, we can partition the variables of the domain into disjoint subsets, build a PDB for each of them and these values can be added and are still admissible. The conditions and applications of disjoint additive PDBs are discussed in [10,6].

In the rest of this paper we will refer to the traditional PDB with a single goal state (i.e., without combining a perimeter) as a *Regular PDB* (R_PDB).

2.4 Perimeter Search

Perimeter Search (PS), a version of bidirectional search, was introduced by Dillenburg and Nelson in [3] and is sketched in figure 3. First, in a preprocessing phase, a breadth-first search from the goal node is performed to a fixed depth d and all the nodes surrounding the goal node at that depth are stored in a hash table. This set of nodes, denoted as P , is called the *perimeter nodes*. It is important to note that throughout this paper, we assume that the initial state is outside the perimeter. This can be assumed if we also store all the nodes around the goal with depth smaller than d (called the *inner nodes*) and whenever we choose to expand one of the stored nodes the search halts.

After the perimeter is built, a forward search from the initial state is performed until one of the nodes in the perimeter is reached. The heuristic function used in PS is the estimation of the distance between a node n and the closest node to it in the perimeter. That is, for a given state n and the set of perimeter states P , the perimeter heuristic

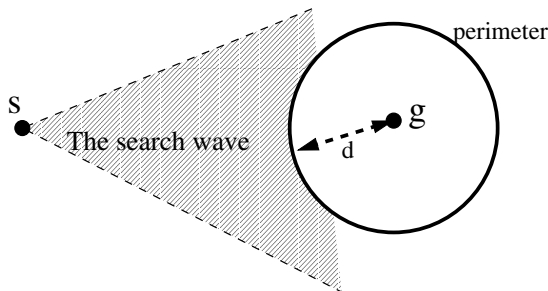


Fig. 3. Perimeter Search

$h_P(n)$ is defined as $h(n) = \min_{p \in P}(h(n, p))$. Adding this amount to $g(n)$ and to the depth of the perimeter results in a higher f -value than a regular search [3]. The reason is as follows. In a regular search $f(n) = g(n) + h(n)$ while in PS $f(n) = g(n) + h_P(n) + d$ where d is the depth of the perimeter. In regular search $g(n)$ is a true value while $h(n)$ is a lower bound. In PS both $g(n)$ and d are true values and only $h_P(n)$ is a lower bound on the true value. Therefore, in PS, a lower bound is given for a smaller portion of the solution path.

The disadvantage of PS is that for each new generated node n we need to perform a large number of heuristic evaluations (from n to all nodes in the perimeter) in order to find the minimum. Thus, there is a tradeoff here. Using larger perimeters might produce higher f -values at the cost of a large number of heuristic calculations for each node. In order to achieve best time performance, [3] show how to calculate the optimal perimeter depth given different characteristics of the domain.

BIDA* - an advanced version of perimeter search was introduced by Manzini [12]. He showed that it is not necessary to perform heuristic calculations between n and all the nodes in P . The minimal heuristic can be found by only calculating the heuristics from node n towards a subset of nodes from the perimeter (called the *active set*).

3 Combining Perimeter Search and Pattern Database Abstractions

Our main new idea in this paper is to combine the usage of a perimeter with heuristics obtained by pattern database abstractions. We present two methods for doing this, a simplified method and an advanced method.

3.1 Simplified Method for Combining Perimeter with PDBs

We would first like to point out the following observation. During the search, a new node n is first matched against all nodes stored in the perimeter table (including inner nodes of depth smaller than d). If the match was negative, then we know that n is outside the perimeter and we can use $d + 1$ as an admissible heuristic even if our regular heuristic is smaller than $d + 1$.

This suggests the *simplified version* for combining PDBs and perimeters [4]. In this *simplified perimeter PDB* (denoted as SP_PDB) we use the regular PDB (R_PDB) as our main heuristic but we also store a table with all the nodes of the perimeter (as well as the inner nodes). The heuristic for node n (outside the perimeter) would be $h(n) = \max(R_PDB(n), d + 1)$. In this version we can see the perimeter as correcting mistaken low heuristic values. For any node n outside the perimeter where $R_PDB(n) \leq d$ then $SP_PDB(n)$ corrects it to $d + 1$.

3.2 Multiple-Goal PDB

Before introducing the advanced method we first describe the notion of multiple-goal PDB. In a regular PDB (R_PDB), we assumed that there is only a single goal state which

² In fact this is applicable for any kind of admissible heuristic and not only to PDBs.

was projected/abstracted to a single goal pattern. The PDB stores distances from all the possible patterns to that specific goal pattern. However, PDBs can be generalized to the case where there are multiple goals. Here we first project/abstract all the goal states and obtain a set of goal patterns. We seed the breadth-first search queue with all these goal patterns (as level 0). We then start the breadth-first search process until the entire pattern space is spanned. The first level that each pattern is reached, is its minimal distance to one of the goal patterns. We call this *Multiple-goal Pattern Database*. *MG_PDB*.

Korf and Felner used such multiple-goal PDB (MG_PDB) to perform half-way search on the 4-peg Towers of Hanoi domain [11]. Due to the symmetry of this domain there exists a given set of possible states which must be in the exact middle of any optimal solution path. Thus, it is sufficient to search and reach one of these states. They built MG_PDB towards this set of *middle states* and obtained a significant reduction in the search effort of a number of orders of magnitude. See [11] for more details.

3.3 Advanced Perimeter Pattern Database (P_PDB)

The nodes of the perimeter can be seen as a set of (multiple) goal states. This further suggests to use MG_PDB where the goal states are the perimeter nodes. First, the preprocessing phase of building the perimeter of depth d is performed. Then we seed the breadth-first search queue of the PDB generator with all the patterns that are abstracted from the nodes of the perimeter³. The resulting PDB will store the minimal distance to any of the patterns abstracted from the perimeter states. We call this *Perimeter PDB* and denote it as (P_PDB). The search is performed using heuristic values from the P_PDB and is halted when it reaches one of the perimeter nodes and chooses it for expansion.

At first sight one might recognize a large potential for speedup in the search when using P_PDB. The main problem of the traditional perimeter search (PS) [3] was that a large number of heuristic calculations must be performed in order to find the closest estimated perimeter node. With P_PDB, however, only a single PDB lookup should be performed in order to get the minimal heuristic towards the perimeter. Surprisingly, as explained and demonstrated below, only limited benefits can be obtained with P_PDB as most of the potential reduction in the search effort is already achieved by the simplified version of SP_PDB.

In formal, P_PDB is built in the following steps:

1. Build a perimeter P of states of depth d around the goal node g in the original search space.
2. Abstract/project each node $p \in P$ to its pattern t_p and seed a breadth-first search queue with all the patterns projected from nodes in P at level 0.
3. Run a breadth-first search in the pattern space until the entire pattern space is spanned. Each new pattern arrived is inserted to a P_PDB. Its level in the breadth-first search corresponds to the distance to the closet pattern that was seeded. This becomes its values on the P_PDB which admissibly estimates the distance to the closest node on the perimeter P .

³ Of course, only the frontier nodes P are seeded and not all the inner nodes.

We now perform the forward search which halts when one of the nodes of the perimeter is chosen for expansion. When a new node n is generated, the following steps are taken in order to calculate its heuristic:

1. We abstract n to its pattern t_n .
2. We look up this pattern in the P_PDB and retrieve the value $P_PDB(t_n)$.
3. The heuristic used for n is $h(n) = P_PDB(t_n) + d$. In fact, we can already add d to the values in P_PDB in the early step of creating the P_PDB. Below, for simplicity, we assume that indeed d was already added to all entries in the P_PDB. Thus, the seeded patterns have a value of d .

4 Analysis of P_PDB

We now provide theoretical analysis of the behavior of P_PDB demonstrated on the (12,4)-TopSpin puzzle. We first would like to make the following definition. Assume that we build a PDB for tokens $\{0, 1, 2, 3, 4, 5\}$. Operators/edges of this state space can be divided to operators that *affect* the tokens of the pattern (i.e., that move some of the tokens of 0, 1, 2, 3, 4, 5) and to operators that do not affect them, (e.g., an operator that moves tokens $\{6, 7, 8, 9, 10, 11\}$). We call them *affecting operators* and *non-affecting operators* respectively.

Next, we analyze the behavior of P_PDB by identifying three cases. Assume that we build a perimeter of depth d as well as the corresponding P_PDB and compare this P_PDB to R_PDB (the regular PDB with no perimeter). Given a state n there are three possible cases for the relation between values obtained by R_PDB and the depth of the perimeter d .

4.1 Case 1: $R_PDB(n) \geq d$

Let n be a node, t_n its abstract/projected pattern and let $R_PDB(t_n)$ and $P_PDB(t_n)$ be its value in R_PDB and P_PDB respectively. Assume that $R_PDB(t_n) \geq d$.

Claim 1: If $R_PDB(t_n) \geq d$ then $P_PDB(t_n) = R_PDB(t_n)$

Proof: Let's call our current pattern t_5 and assume without loss of generality that $R_PDB(t_5) = 5$, $d = 3$ and that $A = \{t_0, t_1, t_2, t_3, t_4, t_5\}$ is the shortest path in the pattern space from the goal pattern t_0 to the current pattern t_5 . Assume also that $\{o_1, o_2, o_3, o_4, o_5\}$ is the sequence of operators (of the pattern space) that travels through A . That is, applying o_1 to t_0 gets to t_1 etc. If we apply $\{o_1, o_2, o_3\}$ to the goal node of the original search space we get to node x which is on the perimeter. It is easy to see that t_3 is the abstract pattern of x . Thus, t_3 is seeded into the breadth-first search queue. Two levels away on the breadth-first search (after applying o_4, o_5) we get to t_5 . Thus, we also have that $P_PDB(t_5) = 5$ (after adding d). Therefore, for states n where $R_PDB(t_n) \geq d$ P_PDB cannot improve on R_PDB since they have the same heuristic values. The same logic applies if $R_PDB(t_n) = d$. Therefore, the only possibility to get improvement by P_PDB is to have higher values for states outside the perimeter whose $R_PDB(t_n) < d$. We have two such possible cases.

4.2 Case 2: $R_PDB(t_n) < d$ and $P_PDB(t_n) = d$

In both case 2 and case 3 $R_PDB(t_n) < d$ but in case 2 $P_PDB(t_n) = d$, that is, t_n will be a projection of a node on the perimeter, while in case 3 $P_PDB(t_n) > d$. Case 2 happens in the following scenario.

Let's call our current pattern t_2 and assume without loss of generality that $R_PDB(t_2) = 2$, $d = 3$ and that $A = \{t_0, t_1, t_2\}$ is the shortest path in the pattern space from the goal pattern t_0 to the current pattern t_2 . Assume also that o_1, o_2 is the sequence of operators (of the pattern space) that travels through A . If we can find a sequence of 3 moves (in the original space) which includes o_1 and o_2 and another non-affecting operator then when we apply that sequence of moves to the goal state (of the original search space) we get to a node x on the perimeter of depth 3 whose abstract pattern is t_2 . Therefore, t_2 is seeded in the queue its value on the P_PDB will be $P_PDB(t_2) = 3$ (as we added the depth to the PDB). As explained below, this is still not enough to obtain an improvement in the search effort over SP_PDB.

4.3 Case 3: $R_PDB(t_n) < d$ and $P_PDB(t_n) > d$

In **case 3**:- the sequence of 3 moves with a non-affecting operator described in case 2 is not applicable in this domain. Thus, t_2 is not on the perimeter of depth 3. The value $P_PDB(t_2)$ will be the first occurrence of t_2 outside the perimeter. Here, we also have two cases:

1. **Case 3.1** t_2 first occurs at depth $d + 1$.
2. **Case 3.2** t_2 first occurs at depth greater than $d + 1$.

The distinction between case 3.1 and case 3.2 will be clear below and we will show that only in case 3.2 a reduction in the search effort can be seen.

4.4 Comparing P_PDB to SP_PDB

We would like to compare SP_PDB to P_PDB. Note that in both cases, we stop the search as soon as we hit the first node on the perimeter. It is easy to see that only in case 3.2 P_PDB can be larger than SP_PDB. That is, only for nodes outside the perimeter with R_PDB value smaller than d and new P_PDB value is at least $d + 2$ we can obtain a higher lower bound than SP_PDB. In the two other cases (2 and 3.1), P_PDB will produce the same final heuristic value as SP_PDB since $d + 1$ is used for nodes outside the perimeter anyway and there is no benefit in using the sophisticated P_PDB unless it provides values of $d + 2$ or higher (for nodes where R_PDB returned values smaller than d). Experimental results below show that case 3.2 is very rare and its contribution is many times rather negligible.

5 Experimental Results for TopSpin

Table [11](#) presents results for (12,4)-TopSpin with a PDB of 9 tokens for perimeters of depths up to 5. Each line corresponds to a different depth of perimeter where the first

Table 1. P_PDB and SP_PDB with 9 tokens on the (12,4)-TopSpin

P-Depth	P-Nodes	P-Patterns	SP_PDB-9	P_PDB-9
0	1	1	3186	3186
1	12	12	2703	2703
2	102	102	2183	2183
3	784	784	1669	1669
4	5,725	5,725	1188	1188
5	39,990	39,990	758	758

Table 2. P_PDB and SP_PDB with 5-tokens on the (12,4)-TopSpin

P-Depth	P-Nodes	Patterns	SP_PDB-5	P_PDB-5
0	1	1	1,226,292	1,226,292
1	12	9	853,617	853,617
2	102	63	517,759	517,759
3	784	637	259,038	259,038
4	5,725	1,688	93,873	93,873
5	39,990	5,112	19,578	19,578

line (perimeter 0) is actually R_PDB. The first column, "P-Nodes", counts the number of nodes at each perimeter depth. The next column, "P-Patterns", presents the number of unique patterns that were abstracted from the perimeter (and were seeded into breadth-first search queue of the P_PDB). Note that for a PDB of size 9 these two numbers are identical since there are no non-affecting operators, i.e., each operator must affect at least one of the 9 tokens of the PDBs. The next two columns give the number of nodes generated by IDA* with SP_PDB and P_PDB respectively. The numbers are the average over 100 random instances.

The SP_PDB column shows that the number of nodes decrease with large perimeters from 3186 with R_PDB to only 758 with SP_PDB of depth 5. It turns out, however, that the results for P_PDB are exactly the same as those of SP_PDB - the two corresponding columns in the table show identical number of generated nodes. Thus, the improvement of P_PDB over R_PDB exclusively belongs to the fact that we stored the perimeter (and used $d + 1$ for heuristics for nodes outside the perimeter that had low R_PDB values). This means that there was not even a single node in our search (of 3.2) where P_PDB was better than SP_PDB. In other words, all patterns with R_PDB smaller than the depth of the perimeter, also occur either at the perimeter or one level away.

We have iterated through the 9-token R_PDB and P_PDB of depth 5 and found the following. There are 6,652,800 different entries in these PDBs. However, in only 375 entries P_PDB had larger values than SP_PDB (case 3.2 above). In 357 of such entries P_PDB had a value of 7 and only 18 entries had value of 8 (In R_PDB these values were smaller than 5 and they were raised to 6 by SP_PDB). We solved all problem instances where the initial states had such values of 8 for P_PDB. For these particular initial states P_PDB did outperform SP_PDB with 28,230 and 29,745 generated nodes on average respectively. This is only a very small reduction.

Table 3. P_PDB and SP_PDB with 9-token on the (17,4)-TopSpin

P-Depth	P-Nodes	Patterns	SP_PDB-9	P_PDB-9
0	1	1	36,525,684	36,525,684
1	17	13	30,379,288	30,379,288
2	187	121	23,963,702	23,963,702
3	1734	990	18,039,530	18,039,530
4	14,841	7,533	12,950,178	12,950,178
5	121,261	54,271	8,783,743	8,783,743

Table 2 presents similar results but this time for a 5-token PDBs for the (12,4)-TopSpin puzzle. Here, for P_PDB the number of patterns abstracted from the perimeter (and seeded in the breadth-first search queue) is smaller than the number of nodes on the perimeter. This is because in this setting there are non-affecting operators, e.g., when the PDB tokens are $\{0, 1, 2, 3, 4\}$ an operator that moves tokens $\{8, 9, 10, 11\}$ is a non-affecting operator. The 5-token P_PDB improves with larger perimeters in a larger factor than the improvement of the 9-token PDB. This is because a heuristic of 5 tokens is weaker and thus the perimeter corrected the heuristics for a larger fraction of nodes. Again, as can be seen in the last two columns P_PDB generated exactly the same number of nodes as SP_PDB.

Table 3 presents results for the 9-token PDB on the (17, 4) – *TopSpin*. The general tendency is the same as the (12,4)-TopSpin with 5-token PDB. Here too, P_PDB was identical to SP_PDB. In both cases the size of a pattern was nearly half the size of the domain. In contrast, in the (12,4)-TopSpin with 9-token PDB the size of the pattern was 3/4 of the domain.

Since P_PDB is never worse than SP_PDB we only use P_PDB in the rest of this paper.

6 Experiments with Multiple PDBs on Rubik’s Cube

We also performed experiments with P_PDB on Rubik’s cube (shown in figure 4) which has about 4×10^{19} different reachable states. There are 20 movable sub-cubes (or cubies) and 6 stable cubies in the center of each face. The movable cubies can be divided into eight corner cubies, with three faces each, and twelve edge cubies, with two faces each. Corner cubies can only move among corner positions, and edge cubies can only move among edge positions. This problem was first solved by Korf in [9]. He used three different PDBs. One PDB for the 8 corners cubies and two PDB of 6 edge cubies each. During the search, all these PDBs are consulted and the maximum value is used as an admissible heuristic.

Our aim is to test influence of P_PDB when maxing a number of PDBs. We used the same PDB abstractions for Rubik’s cube and built P_PDBs for perimeters of size 0 to 6.

Table 4 presents the results of these experiments. Each row corresponds to a different perimeter. The first column presents the number of Rubik’s cube nodes on each perimeter. The next two columns present the number of unique corner and edges patterns that were seeded in the breadth-first search queue that built the P_PDB.

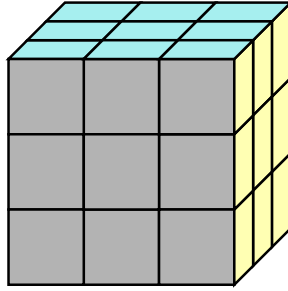


Fig. 4. $3 \times 3 \times 3$ Rubik's cube

Table 4. P_PDB for Rubik's cube

Perimeter	P-nodes	P-corners	P-edges	Corners P_PDB	All P_PDBs
0	1	1	1	6,847,015,626	42,926,347
1	18	18	16	5,310,262,275	42,926,311
2	243	243	197	3,606,961,019	42,926,288
3	3,240	2,874	2,400	2,085,111,540	42,926,166
4	43,239	29,911	27,717	983,789,791	42,924,315
5	574,908	235,049	286,529	377,920,023	42,858,624
6	7,618,438	1,404,636	2,452,120	119,100,311	41,366,838

The last two rows present the number of generated nodes when we solved the problem with IDA* using P_PDB as the heuristic. The "Corners P_PDB" column is the case when we only used the corners P_PDB while the "All P_PDBs" column is the case where all three P_PDBs were used and we took their maximum as the heuristic. All data numbers in these columns are average over 100 instances that were generated by 14 random moves.

As can be seen, when using only a single abstraction (of corners), the usage of P_PDB with perimeter of depth 6 reduced the number of generated nodes over R_PDB by up to a factor of 57. However, when using all three PDB's then the effect of using P_PDB is almost insignificant.

The explanation for this is that a single PDB may be very inaccurate in many cases. For example, assume that a state is 6 moves away from the goal state but all the corner cubies are in their goal location. Thus, when using only the regular corners PDB (without the perimeter) we will get a heuristic of 0. Adding any other information to correct this misleading heuristic is beneficial. This can be done either by introducing more PDBs (such as the edges PDBs) or by using a P_PDB. If we also query the edges PDBs (in addition to querying the corner PDB) then the edges PDB heuristics will reveal that the edges are rather far away from their goal. Alternatively, if we use a perimeter, the P_PDB and the perimeter will reveal that we are outside the perimeter.

It turns out that for the special case of Rubik's cube, adding two more PDBs is more beneficial than adding a perimeter and using a P_PDB.

7 Combining Perimeter with Disjoint PDBs

One of the classic examples in the AI literature of a single-agent path-finding problem is the sliding-tile puzzle. It consist of a square frame containing a set of numbered square tiles, and an empty position called the blank. The legal operators are to slide any tile that is horizontally or vertically adjacent to the blank into the blank position. The problem is to rearrange the tiles from some random initial configuration into a particular desired goal configuration. The 4×4 15-puzzle contains about 10^{13} reachable states.

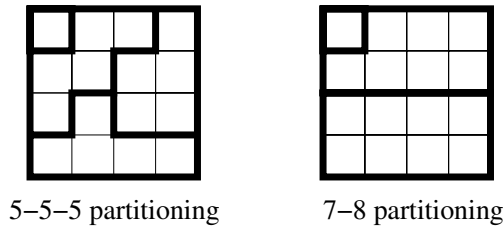


Fig. 5. The 5-5-5 and 7-8 disjoint partitioning of the 15 puzzles

Additive disjoint pattern databases provide the current best admissible heuristic for the sliding-tile puzzles [6,10]. The tiles are partitioned into disjoint sets (patterns) of tiles and a PDB is built for each set. The PDB stores the cost of moving the tiles in the given subproblem from any given arrangement to their goal positions. If for each set of tiles only the moves of tiles from the given set are counted, then values from different disjoint PDBs can be added and the result is still admissible. An $x - y - z$ partitioning is a partition of the tiles into disjoint sets with cardinalities of x, y and z . Figure 5 presents the $5 - 5 - 5$ and $7 - 8$ partitionings for the 15-puzzle which were first used in [6].

The benefit of using disjoint additive P_PDBs is limited. This is because the following reason. Let's assume that we have two disjoint P_PDBs PDB_a and PDB_b which for a given node, n , return heuristic values of 1 and 2 respectively. Thus, we take $h = 1 + 2 + d$ (and not $h = 1 + 2 + 2d$) as an admissible heuristic. Here we cannot add d to the values of all the PDBs because d must be added only once. However, we might lose information here. Assume that $PDB_a(n) = 1$ because the tiles of PDB_a in S are very close to node P_a from the perimeter and that $PDB_b(n) = 1$ because tiles of PDB_b in n are close to node P_b from the perimeter. The optimal solution path will only reach **one node** at the perimeter while we got values from two different perimeter nodes.

A trivial case for that would be a set of 15 disjoint P_PDBs, one for each tile (simulating Manhattan distance). Now, assume that tile 3 is in location 5. There might be a node on the perimeter where tile 3 is also in location 5. Thus, for tile 3 we get a value of 0. If the perimeter is large enough we might get 0 for all the tiles. Therefore, the idea of P_PDB is only limited to small perimeters where disjoint PDBs are used.

Table 5 presents results for the 5-5-5 PDB and 7-8 PDB disjoint PDBs of the 15 puzzle. Each line corresponds to a different depth of perimeter. No symmetries or reflections were taken here. The different rows correspond to different depth of perimeters.

Table 5. P_PDB for disjoint partitioning of the 15 puzzle

Perimeter	Nodes	Seconds	heuristic
5-5-5 PDB			
0	16,826,097	3.60	39.72
1	8,551,956	1.82	40.04
2	7,002,163	1.49	40.34
3	12,499,815	2.66	39.55
4	30,687,664	6.54	38.45
7-8 PDB			
0	136,289	0.20	44.75
1	94,028	0.12	45.04
2	179,533	0.23	44.26
3	269,284	0.35	43.66

The two middle columns present the average number of generated nodes and the average time in seconds of solving the set of the random 1000 initial states.

The table shows that the P_PDB idea is only beneficial for small perimeter depths. The best variation is a perimeter of depth 2 for the 5-5-5 PDB and of depth 1 for the 7-8 PDB. With larger perimeters, the heuristics become less accurate and the number of nodes increase.

8 Summary and Future Work

We studied two methods for combining perimeter search and pattern databases - the simplified version, SP_PDB, and advanced version, P_PDB. At first glance a great potential might be seen for P_PDB. In practice, however, only limited (if any) improvement and reduction in the search effort of P_PDB was seen over SP_PDB. In this paper we explained why this is the case. Only for cases, where the heuristic value is increased from inside the perimeter (in R_PDB) to outside by more than 1 (by P_PDB), as in case 3.2 above, a benefit can be achieved and this only happens rarely. Furthermore, advance methods for using PDBs such as maxing and adding a number of PDBs are either competing with the perimeter for additional information (maxing) or provide more difficulties for using a perimeter (adding) and thus limited benefits are seen when using a perimeter in conjunction with these methods. However, for a single PDB using a perimeter might be useful.

A related idea is called *partial PDB*. Here, one builds a PDB for an abstract problem but runs the PDB generator only up to a certain depth d . During the search, we consult the PDB and if the corresponding pattern does not have an entry in the PDB, we can use $d + 1$ as the heuristic. This idea is independently studied by [11].

Another question which needs to be addresses is the memory requirements. Storing a perimeter consumes memory and given a fixed amount of memory there is a competition between the perimeter table and the PDB over the available memory. Future research can study the tradeoff between the sizes of these two lookup tables.

Acknowledgements

This research was supported by the Israel Science Foundation (ISF) under grant number 728/06 to Ariel Felner.

References

1. Anderson, K., Schaeffer, J., Holte, R.: Partial pattern databases. In: SARA-07 (to appear)
2. Cullberson, J.C., Schaeffer, J.: Pattern databases. *Computational Intelligence* 14(3), 318–334 (1998)
3. Dillenburg, J.F., Nelson, P.C.: Perimeter search. *Artificial Intelligence* 65, 165–178 (1994)
4. Edelkamp, S.: Planning with pattern databases. In: *Proceedings of the 6th European Conference on Planning (ECP-01)*, pp. 13–34 (2001)
5. Edelkamp, S.: Symbolic pattern databases in heuristic search planning. In: *International Conference on AI Planning and Scheduling (AIPS)*, pp. 274–293 (2002)
6. Felner, A., Korf, R.E., Hanan, S.: Additive pattern database heuristics. *Journal of Artificial Intelligence Research (JAIR)* 22, 279–318 (2004)
7. Holte, R.C., Felner, A., Newton, J., Meshulam, R., Furcy, D.: Maximizing over multiple pattern databases speeds up heuristic search. *Artificial Intelligence* 170, 1123–1136 (2006)
8. Holte, R.C., Newton, J., Felner, A., Meshulam, R., Furcy, D.: Multiple pattern databases. In: *ICAPS*, pp. 122–131 (2004)
9. Korf, R.E.: Finding optimal solutions to Rubik’s Cube using pattern databases. In: *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pp. 700–705 (1997)
10. Korf, R.E., Felner, A.: Disjoint pattern database heuristics. *Artificial Intelligence* 134, 9–22 (2002)
11. Korf, R.E., Felner, A.: Recent progress in heuristic search: A case study of the four-peg towers of hanoi problem. In: *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pp. 2324–2329 (2007)
12. Manzini, G.: BIDA*: an improved perimeter search algorithm. *Artificial Intelligence* 75, 347–360 (1995)
13. Zhou, R., Hansen, E.: Space-efficient memory-based heuristics. In: *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-04)*, pp. 677–682 (2004)
14. Zhou, R., Hansen, E.: Structured duplicate detection in external-memory graph search. In: *Proc. AAAI-04*, pp. 683–689 (2004)

Solving Satisfiability in Ground Logic with Equality by Efficient Conversion to Propositional Logic

Igor Gammer and Eyal Amir

Department of Computer Science
University of Illinois, Urbana-Champaign
{igammer2, eyal}@uiuc.edu

Abstract. *Ground Logic with Equality* ($GL^=$) is a subset of First-Order Logic (FOL) in which functions or quantifiers are excluded, but equality is preserved. We argue about $GL^=$'s unique position (in terms of expressiveness and ease of decidability) between FOL and Propositional Logic (PL). We aim to solve satisfiability (SAT) problems formulated in $GL^=$ by converting them into PL using a satisfiability-preserving conversion algorithms, and running a general SAT solver on the resulting PL Knowledge Base (KB). We introduce two conversion algorithms, with the latter utilizing the former as a subroutine, and prove their correctness - that is, that the translation preserves satisfiability. The main contribution of this work is in utilizing input fragmentation to yield PL KBs that are smaller than possible prior to our work, thus resulting in the ability to solve $GL^=$ SAT problems faster than was possible before.

1 Introduction

Satisfiability in PL is well-studied in Computer Science because of its theoretical significance and the multitude of practical applications that it has. A significant number of real-world problems can be modeled in or translated to propositional knowledge bases (KBs), such that solution to SAT on the PL KB can be easily translated to a solution of the original problem. As a consequence, modern-day PL SAT solvers operate very efficiently in many practical cases. Although the general problem of satisfiability in PL is NP-complete in the number of variables, the best practical SAT solvers applied to moderately-sized problems can terminate in minutes and hours, not years.

However, PL is not a sufficiently complex language for some problems. It is generally well understood that every representation language faces a trade-off between expressive power (which determines the subset of real-world applications that can be reasonably modeled in that language) and complexity (which determines, among other, the running time of the algorithms performing inference in that language). For example, PL has no notion of equality, a fundamental notion which is required in many applications, such as game playing (and specifically game playing with partial information such as Kriegspiel [1]), where one needs to assert and reason that two positions or game pieces are equal.

A natural solution is to use a language with more expressive power, and the usual choice for many applications is FOL. The price of much higher expressive power, however, is complexity of algorithms; even though reasoning algorithms for FOL exist ([2] is one example), they are less efficient than PL SAT solvers. Even assuming the best progress in improving the algorithms, theoretical results challenge for FOL inference as FOL, unlike PL, is only semi-decidable.

$GL^=$ can be seen as a compromise between complexity and expressive power. Many problems that require equality do not, however, need the full expressiveness of FOL and can be well-expressed in $GL^=$. Having strictly more expressive power than PL (any PL KB can be easily translated to an equivalent KB in $GL^=$ by choosing a constant for each proposition in L of PL; choosing no predicates; and fixing $\{true, false\}$ as the domain) but still being sufficiently simple to permit efficient inference, $GL^=$ establishes itself at a favorable level of formalism.

In this paper, we explore the inference in $GL^=$, and particularly solving SAT problems. Rather than introducing a brand-new SAT solving routine in $GL^=$, we will investigate algorithms for translating arbitrary KBs in $GL^=$ into "equivalent" KBs in PL, where equivalence is defined as preserving satisfiability. That is, the resulting KB in PL will be satisfiable if and only if the original KB in $GL^=$ was satisfiable. Having an efficient algorithm for performing such translation easily leads us to solving SAT in $GL^=$ by utilizing highly efficient, off-the-shelf PL SAT solvers. This approach has an added benefit of automatically improving as SAT solvers improve, and thus can be seen as gaining power "for free" as the field of SAT solving progresses (whereas a new $GL^=$ SAT algorithm would have to be manually improved to include any new ideas the SAT solving community introduces).

To accomplish the conversion, we first describe a "naive" encoding of $GL^=$ KBs into PL. The encoding is naive only in the sense of being natural, and thus not very efficient; its correctness, however, is proved. The last section describes the main contribution of this paper: an advanced encoding algorithm that employs divide-and-conquer approach while using the naive encoding as a subprocedure. Our proposed algorithm has an added advantage of being independent of the actual encoding subprocedure used, so long as that subprocedure is satisfiability-equivalent (in the sense defined later). As such, our algorithm will benefit from any improvement to the underlying procedure without any explicit change being necessary.

The rest of the paper is organized as follows. In Section 2, we give a brief overview of fundamentals associated with the topic at hand. Section 3 proposes a simple yet effective conversion algorithm, proves its correctness, and then analyzes its efficiency. Section 4, which form the core of this paper, introduces the more efficient algorithm mentioned above, which functions by fragmenting the input and using the previously formulated algorithm on the fragments. We prove that this algorithm is correct - that is, even though some information appears to be missing from the encoding (which fact is chiefly responsible for the lower encoding size and thus higher SAT solving efficiency), satisfiability is preserved. Finally, Section 6 presents an overview of related work, examines our future plans, and provides a summary for this paper.

2 Fundamentals

2.1 Ground Logic with Equality

Ground Logic with Equality ($GL^=$) is a subset of FOL which excludes quantifiers and functions, but keeps constants and predicates, as well as equality. The only other element of FOL - variables - is not specifically excluded, but becomes synonymous with constants due to the exclusion of quantifiers. No additional restrictions are placed on the formulas or KBs.

In this section, we give formal definitions necessary to establish a framework within which the discussion of the rest of this paper will occur. All of these definitions are adapted from the corresponding definitions of FOL, with changes required to accommodate the lack of functions and quantifiers. In particular, it is worthwhile to mention the basic terms of FOL which are missing from $GL^=$.

Besides variables, three other terms of FOL are no longer applicable: free occurrence, sentences and terms. In FOL, a variable is said to occur free if it is not bound by a quantifier; since we don't have quantifiers, we don't distinguish between free and bound occurrences. A sentence is a special case of formula which contains no free variables; in $GL^=$, every formula is a sentence. Finally, a term of FOL is an entity which is interpreted as an element in the universe - that is, either a ground term (variable or constant), or a function symbol applied to terms. Because we do not have functions (or variables), our terms will always be constant symbols, and thus do not deserve a definition of their own.

We will use the standard definitions from formal logic, with their standard meanings. The same terms applied to $GL^=$ will be assumed to hold naturally derived meaning as well. We will talk about *languages*, *formulas* (and sets of formulas, which we will denote as *Knowledge Bases (KBs)*), *interpretations* and *models*, as well as *satisfiability* and *unsatisfiability*. The most important definitions are reproduced below.

Definition 1 (Syntax). *A formula in $GL^=$ with language L is obtained in one of the six following ways:*

- a. $x = y$, where x and y are constants.
- b. $P^k x_1 \dots x_k$, where P^k is a k -ary predicate in L , and for each i , x_i is a constant in L .
- c. $F_1 \vee F_2$, where F_1 and F_2 are formulas.
- d. $F_1 \wedge F_2$, where F_1 and F_2 are formulas.
- e. $\neg F$, where F is a formula.
- f. (F) , where F is a formula.

Definition 2 (Interpretation). *An interpretation in $GL^=$ with language L is a pair (A, β) , where*

- a. A is an arbitrary set, also referred to as domain or universe.
- b. β is a map (also referred to as an assignment) on L , defined as follows:
 - (a) For each constant x from L , β assigns an element of A .
 - (b) For each k -ary predicate P from L , β assigns a subset of A^k .

Definition 3 (Semantics). Let F be a formula in $GL^=$ with language L , and let $I = (A, \beta)$ be an interpretation on L . Then F is said to **hold** (alternatively, to **be true**) under I if and only if:

- a. $x = y: \beta(x) = \beta(y)$; that is, the assignment β maps x and y into the same element of the domain A .
- b. $P^k x_1 \dots x_k: (\beta(x_1), \dots, \beta(x_k)) \in \beta(P^k)$; that is, the k -tuple obtained by mapping each of the predicate arguments is an element of the subset (of all k -tuples of the domain) to which the assignment maps the predicate.
- c. The remaining cases are obvious and are omitted here for space.

If F does not hold, it is also said to **be false**.

3 Naive Conversion

In this section, we explore the naive conversion of a $GL^=$ KB into a PL KB. Note that "naiveness" applies only to efficiency; the correctness of the conversion is preserved. We first describe the algorithm and then state and prove the correctness theorem.

3.1 Algorithm

The basic premise of the conversion is to associate a proposition with every *instance* of a predicate or equality. This in particular means that the original KB determines the language of the resulting PL KB. The increased expressive power of $GL^=$ compared to PL is responsible for this increase in complexity.

Our final goal is to find an efficient (that is, generating as small an output as possible) conversion that preserves satisfiability. We define this precisely.

Definition 4 (Conversion Algorithm). A conversion algorithm is one that accepts as input a KB in $GL^=$ and outputs a KB in PL. Alternatively, a conversion function is one with domain of all KBs in $GL^=$ and codomain of all KBs in PL.

Definition 5 (Satisfiability Equivalence for Formulas and Algorithms). We define satisfiability equivalence for both formulas and algorithms.

- a. Given a $GL^=$ KB Φ and a PL KB Φ' , we call Φ and Φ' **satisfiability equivalent**, or **SAT-EQ**, if Φ and Φ' are either both satisfiable, or both unsatisfiable.
- b. A conversion algorithm A with the property that, for every $GL^=$ KB Φ , whenever $A(\Phi) = \Phi'$, then Φ and Φ' are satisfiability equivalent, is satisfiability equivalent.

We define the translation of a single formula first.

Definition 6 (Naive Translation). Let ϕ be a formula in $GL^=$ with language L . The Naive Translation of ϕ , $NT(\phi)$, is a formula ϕ' in PL with language L' defined on the structure of ϕ as follows:

- a. $x = y$: A new proposition \overline{EXY} , which is created and added to L' if it is not already there.
- b. $P^k x_1 \dots x_k$: A new proposition $\overline{PX_1 \dots X_k}$, which is created and added to L' if it is not already there.
- c. $F_1 \vee F_2$: $NT(F_1) \vee NT(F_2)$.
- d. $F_1 \wedge F_2$: $NT(F_1) \wedge NT(F_2)$.
- e. $\neg F$: $\neg NT(F)$.
- f. (F) : $(NT(F))$.

As defined above, L' contains a unique proposition for each instance of equality and for each instance of predicate in ϕ .

If the original formula does not contain equality, this translation is indeed a satisfiability-equivalent algorithm. However, as soon as equality is introduced, this property no longer holds. To illustrate this, consider a simple counterexample with a $GL^=$ KB whose language contains one binary predicate and three constants, consisting of three formulas: $\{Pxy, \neg Pxz, y = z\}$. Clearly, this KB is UNSAT (any interpretation of this KB would have to map y and z to the same element in its domain A , and then the pairs (x, y) and (x, z) would have to be mapped to the same pair in A^2 ; but that pair cannot both belong and not belong to a subset of A^2 to which the interpretation maps P). However, Naive Translation of this KB produces a PL KB containing three propositions $\{\overline{PXY}, \neg \overline{PXZ}, \overline{EYZ}\}$ with no additional constraints between them. This KB is satisfiable, for example by an assignment $\{\overline{PXY} \rightarrow \text{true}, \overline{PXZ} \rightarrow \text{false}, \overline{EYZ} \rightarrow \text{true}\}$.

In order to preserve satisfiability, we need to encode additional constraints interpreting equality in propositional logic. In general, there are three ways of dealing with equality in inference ([3]): adding additional formulas, using special rules (such as demodulation and paramodulation), and modifying the inference rule to be informed of equality (such as by introducing superposition into rule calculus). While the last two ways are useful for general inference, it is not immediately clear how to implement either of them for conversion into PL. Therefore, we will explicitly add formulas that capture the precise meaning of equality.

Definition 7 (Equality Semantics). Let L be a $GL^=$ language. Then the Equality Semantics (ES) of L is a set of $GL^=$ formulas defined as the union of:

- a. For every constant x in L : $x = x$.
- b. For every pair of constants x, y in L : $x = y \rightarrow y = x$.
- c. For every triple of constants x, y , and z in L : $(x = y \wedge y = z) \rightarrow (x = z)$
- d. For every k -ary predicate P^k and for every $2k$ -tuple of variables $(a_1, \dots, a_k, b_1, \dots, b_k)$ in L :

$$((a_1 = b_1) \wedge \dots \wedge (a_k = b_k)) \rightarrow (P^k a_1 \dots a_k \leftrightarrow P^k b_1 \dots b_k) \quad (1)$$

Note that the formulas which capture the notion of equality do not depend on the contents of our KB, but only on the language. Nor are these formulas part of the KB (although adding them will not affect satisfiability); instead, they will be translated to PL using the mechanism defined above (NT) and added to the PL KB.

Definition 8 (Naive Conversion). *Let Φ be a KB in $GL^=$ with language L . The Naive Conversion of Φ , $NC(\Phi)$, is a KB Φ' in PL defined as:*

$$\Phi' := NT(\Phi) \cup NT(ES(L)) \quad (2)$$

It is instructive to revisit the earlier counterexample and ensure that it no longer holds, which we will not do for space reasons.

3.2 Correctness

We now state and prove the main result of this section: the conversion defined above preserves satisfiability. The proof will be an immediate application of two analogous results. Given an interpretation in either $GL^=$ or PL, it is possible to construct an interpretation in PL (resp. $GL^=$) for which an important property holds: if the original interpretation was a model for any KB in $GL^=$ (resp. PL), then the constructed interpretation will also be a model for a KB in PL (resp. $GL^=$) related to the original interpretation by NC.

Theorem 9 (Naive Conversion is SAT-EQ). *Let Φ be a KB in $GL^=$ with language L , and let Φ' be $NC(\Phi)$. Then Φ is satisfiable if and only if Φ' is satisfiable. That is, NC is SAT-EQ.*

Proof. \square The proof has two parts, each of which is further subdivided into two sections. In first part, we first prove that $SAT \Phi \rightarrow SAT \Phi'$, and in the second part we prove the converse. Within each part, our proof will proceed as follows: Given that a KB is SAT, there must exist an interpretation under which it holds. We then (a) construct an interpretation for the other KB and (b) show that the other KB holds under that interpretation. For space reasons, we only show the construction in each case.

[SAT $\Phi \rightarrow$ SAT Φ'] Assume that Φ is satisfiable. Then there exists an interpretation $I := (A, \beta)$ under which Φ holds. We will construct the PL assignment β' and show that Φ' holds under this assignment. We define β' as follows:

- a. For every proposition \overline{EXY} of the first kind, β' assigns *true* if $\beta(x) = \beta(y)$, and *false* otherwise.
- b. For every proposition $\overline{PX_1 \dots X_k}$ of the second kind, β' assigns *true* if $(\beta(x_1), \dots, \beta(x_k)) \in \beta(P)$, and *false* otherwise.

It is now possible to show that Φ' holds under β' ; the precise argument is omitted for space.

¹ Due to space concerns, we omit some proofs, and only show the brief outlines of the others. Full versions of all proofs of this paper are available from the authors.

[**SAT** $\Phi' \rightarrow$ **SAT** Φ] Assume that Φ' is satisfiable. Then there exists an assignment β' under which Φ' holds. We will construct the $GL^=$ interpretation $I = (A, \beta)$ and show that Φ holds under this interpretation.

To define A , we need an intermediate result that will later help in our proof.

Lemma 10. *Let L_{const} be the subset of L containing precisely all constants of L . Let R be a binary relation defined on L_{const} as follows: for any constants a and b from L_{const} , $R(a, b)$ holds if and only if $\beta'(\overline{EAB}) = true$. Then R is an equivalence relation.*

Proof. Omitted for space; a simple argument showing reflexivity, symmetry and transitivity directly suffices. \square

As an equivalence relation, R partitions L_{const} into equivalence classes. Let N be the total number of those classes (where $1 \leq N \leq |L_{const}|$). We define the domain A to be $\{1, 2, \dots, N\}$, the set of positive integers from 1 to N , inclusive. Further, we associate with every equivalence class a unique number between 1 and N in any deterministic way. We now define β on L_{const} as follows: for every constant a , β assigns to a the unique number corresponding to the (unique) equivalence class to which a belongs. Note that this is well-defined, because of the properties of equivalence classes (namely, that a belongs to precisely one equivalence class).

It remains to define β for predicates. Let P be a k -ary predicate of L . Then we define β as follows:

$$\beta(P) := \{(\beta(a_1), \dots, \beta(a_k)) \mid \beta'(\overline{PA_1 \dots A_k}) = true\} \quad (3)$$

That is, β defines P to contain precisely all those k -tuples of elements of A for which the corresponding predicate of L' is assigned *true* by β' .

This completes the construction of interpretation $I = (A, \beta)$ for L . It remains to show that I is a model of Φ ; this is omitted for space. \square

We conclude this subsection by noticing two results that we have used in the proof above, and extracting them to form separate lemmas. Both of these results will be useful further in discussing Craig's Interpolation applications. The correctness of both lemmas follows easily from the generality (independence of particular Φ and Φ' , other than the relation $\Phi' = NC(\Phi)$ connecting them) of construction and proof of the respective part of Theorem 9; the proofs are omitted for space.

Lemma 11. *Let $I := (A, \beta)$ be a $GL^=$ interpretation. Then it is possible to construct a PL assignment β' such that, for any $KB \Psi$ in $GL^=$, if I is a model of Ψ , then β' is a model of $NC(\Psi)$.*

Lemma 12. *Let β' be a PL assignment. Then it is possible to construct a $GL^=$ interpretation $I := (A, \beta)$ such that, for any $KB \Psi$ in $GL^=$, if β' is a model of $NC(\Psi)$, then I is a model of Ψ .*

3.3 Analysis

The theorem's statement and proof together form the first complete method by which we can reach the goal of this paper: given a KB in $GL^=$, construct a SAT-EQ KB in PL. While the method is correct, it is not optimal. Recall that our motivation was the ability to solve SAT in $GL^=$ using off-the-shelves PL SAT solvers. Such solvers vary in efficiency, but invariably depend in the execution time on the number of propositions in the PL KB. Thus, a measure of the efficiency of any conversion algorithm can be presented by computing the number of PL propositions this algorithm generates, as a function of its input.

Definition 13. *Let C be a conversion algorithm. The size of C , denoted $Size(C)$, is a function which accepts the number of predicates m and the number of constants n , and outputs the number of propositions which C generates from an input having m predicates and n constants.*

Suppose that the input $GL^=$ KB contains n constants and m predicates. We wish to estimate $Size(NC)$; that is, rather than arriving at a precise number, we wish to obtain a complexity bound for the number of propositions generated. By examining the construction described above (with detailed treatment omitted for space), we can conclude that the upper bound on $Size(NC)$ is $n^2 + mn^k$. In a particular case of having only unary and binary predicates, this bound can be further simplified to $O(mn^2)$.

4 Advanced Conversion by Partitioning

The previous section presented NC, a method to encode arbitrary $GL^=$ KBs, and proved its correctness. Having analyzed the performance of NC, we will now attempt to devise a more efficient procedure which uses the presented method as a subroutine.

4.1 Motivation

While NC is correct, it generates a propositional encoding with size of $O(n^2 + mn^k)$. Since the size of a conversion algorithm is defined as the number of propositions it can generate, and since modern SAT solvers are designed so that their efficiency depends greatly on the number of propositions in the input, we would like to decrease the size of a conversion method while still ensuring its correctness. In the worst case, a SAT solver will take time exponential² in the number of propositions, and hence, for NC, exponential in both n and m and super-exponential in k ³.

² Provided, of course, that $P \neq NP$.

³ The latter is not as bad as it may seem, because typically k is bounded by a very small number (in fact, the case of k bounded by 2 - that is, the case of having only unary and binary predicates - is of importance by itself. For the case of k bounded by 1, and thus having only unary predicates, see [4]). However, the former may act as a severely limiting factor, since either n or m , or both, can be large in practical applications.

The above, however, is insufficient motivation for devising a better method, which would use NC as a subprocedure, since we have so far only enumerated the deficiencies of NC. The key observation that allows a better method to be devised is that NC produces some extraneous propositions - those that are not necessary to propagate the semantics of satisfiability. We will attempt to remove as many as possible of those extraneous propositions while not violating the satisfiability equivalence of our conversion algorithm.

4.2 Description

The concept of dividing the input into parts and applying some transformation to those parts has long been among the standard tools for fighting complexity; see, for example, [5] and [6]. The principal notion is that, even if the underlying transformation is still exponential in the size of its input, applying that transformation to each of the partitions dramatically decreases that "size of its input". We will adapt a simple algorithm: Given an input $GL^= KB$, instead of applying a conversion procedure⁴ to that entire KB, we will instead partition that KB into two fragments, and apply the conversion procedure to each in turn. The result, then, will be the union of the obtained results.

The primary potential concern is the possible loss of information resulting from severing the connections between entities which are assigned to different partitions. Intuitively, it might seem that such an algorithm will not provide for a satisfiability-equivalent conversion, because some information will be lost - specifically, it might be that while the input KB itself was unsatisfiable, the two segments are satisfiable by themselves, and that their translations will also be satisfiable, so that the union of the translations will fail to preserve the unsatisfiability of the original KB. However, this reasoning fails to account for the extra information generated by NC - specifically, that generated by equivalence semantics processing. Indeed, in what follows we will show that the mere fragmentation, without any extra reasoning (other than the conversion procedure) being performed on either of the fragments, is sufficient to maintain satisfiability equivalences.

We conclude this subsection by formalizing our algorithm and stating (but not yet proving) the associated correctness theorem.

Our precise algorithm for converting a $GL^= KB$ will thus be as follows:

- a. Given a $GL^= KB \Phi$:
- b. Separate Φ into two fragments, $NC(\Phi_1)$ and $NC(\Phi_2)$ ⁵.

⁴ We will be using NC (the only conversion procedure we have so far presented), but notice that this algorithm does not depend on the choice of a particular conversion procedure. Indeed, this is an explicit strength of this algorithm (and of the divide-and-conquer algorithms in general), because any improvements to the underlying conversion procedure will be benefited from without any need to change our algorithm, that is, "for free".

⁵ We leave the precise way of doing this unspecified for now. In the case of two fragments, which is being considered here, any fragmentation is sufficient.

- c. Run the conversion algorithm on the first fragment.
- d. Run the conversion algorithm on the second fragment.
- e. Join the results.

Thus, instead of computing $NC(\Phi)$, we will be computing $NC(\Phi_1) \cup NC(\Phi_2)$. We need, of course, to show that this is satisfiability-equivalent. We thus formulate the result, which we hold to be the most important single contribution of this paper, and which we will prove in the following subsection.

Theorem 14. *Let Φ be a $GL^=$ KB partitioned into Φ_1 and Φ_2 . Let Φ' be a PL KB obtained as follows:*

$$\Phi' := NC(\Phi_1) \cup NC(\Phi_2) \tag{4}$$

Then, Φ and Φ' are satisfiability-equivalent (Definition 5). Equivalently, a conversion algorithm that applies 4 is satisfiability-equivalent.

We will develop the proof in the following subsections, after having introduced the tools that we will use.

4.3 Craig's Interpolation

Craig's theorem, first published in 7, is a classical result that forms one of the bases of many "divide-and-conquer" approaches in FOL and its fragments. The original result has been extended several times; however, the basic version will be satisfactory for our purposes. The key property, which will be useful for us in reducing the size of the conversion (our entire reason for even attempting to find a conversion method different from the original NC), is that this message will have a potentially greatly reduced language compared to that of each fragment. The Craig's interpolants, thus, will have only those predicates and constants which appear in both fragments.

We state the form we will use here for future reference.

Proposition 15. *Let Φ_1 and Φ_2 be two $GL^=$ KBs. Then there exists a $GL^=$ KB Φ , referred to as a **Craig's interpolant**, with the following properties:*

- a. $\Phi_1 \models \Phi$.
- b. If Φ_1 is inconsistent with Φ_2 , then Φ is also inconsistent with Φ_2 .
- c. The language of Φ is the intersection of the languages of Φ_1 and Φ_2 . That is, Φ contains only those predicates and only those constants that appear in both Φ_1 and Φ_2 .

Proof. Craig's Interpolation for FOL is discussed in great detail in 7. The only concern here is that, since the original result is formulated for FOL, we need to ensure that the interpolants of $GL^=$ formulas can be expressed in $GL^=$; that is, that the interpolants will not contain quantifiers so long as the input itself does not. This is addressed in 8. □

4.4 Craig's Interpolation and Partitioning

We will state and prove some auxiliary results first. The following result, which postulates that NC respects entailment, forms the core of our argument.

Lemma 16. *Let Φ_1 and Φ_2 be $GL^=$ KBs. If $\Phi_1 \models \Phi_2$, then $NC(\Phi_1) \models NC(\Phi_2)$.*

Proof. Let β be an arbitrary assignment that satisfies $NC(\Phi_1)$. We need to show that β also satisfies $NC(\Phi_2)$. Let $M = (A, \alpha)$ be a $GL^=$ interpretation constructed from β using the process described in the second half of Theorem 9. By Lemma 12, since $\beta \models NC(\Phi_1)$, $M \models \Phi_1$. By hypothesis, $\Phi_1 \models \Phi_2$, so $M \models \Phi_2$ as well. Let β' be a PL assignment constructed from M using the process described in the first half of Theorem 9. By Lemma 11, since $M \models \Phi_2$, $\beta' \models NC(\Phi_2)$.

We will now show that β and β' agree on $L(NC(\Phi_2))$; that is, that for every proposition in the language of $NC(\Phi_2)$, either both β and β' assign *true*, or both assign *false*. Intuitively, this is not at all surprising, for even though β was an arbitrary assignment, it is related to β' in that the latter was created from M which in turn was created from the former. If both constructions are reasonable, it is not unnatural to expect that they are "reversible" in some sense, and that β and β' will indeed agree on all of the propositions we are interested in.

Because $NC(\Phi_2)$ was created by applying NC , as described in Definition 8, the only propositions its language may contain are those created by NT , as described in Definition 6. Specifically, it may only contain propositions of two forms, which we treat in order. For both forms, we consider the only two possibilities for the truth value β' assigns to that proposition, and for each such possibility we "unroll" the two construction, first deducing what must have been true about M for β' to obtain the value that it has; and then deducing what must have been true about β for M to obtain the value that we have deduced it must have. In all cases, we will show that the truth value that β assigns to the proposition in question must be the same as that assigned by β' .

a. Propositions of form \overline{EAB} :

- (a) If β' assigns *true* to \overline{EAB} , then (by construction of β' from M as described by the first point of the first half of Theorem 9), it must have been the case that $\alpha(a) = \alpha(b)$. But then, because α was constructed from β by the procedure described by the second half of Theorem 9, it must have been the case that a and b were in the same equivalence class with respect to the binary relation R defined in Lemma 10, so $R(a, b)$ must have held. However, because of the way in which R was defined, this requires that β must have assigned *true* to \overline{EAB} . Thus, in this case, β and β' indeed agree on the proposition in question.
- (b) If β' assigns *false* to \overline{EAB} , then the argument is similar to the one above, and is omitted for space.

b. Propositions of form $\overline{PX_1 \dots X_k}$:

- (a) If β' assigns *true* to $\overline{PX_1 \dots X_k}$, then (by construction of β' from M as described by the second point of the first half of Theorem 9), it must have

been the case that the k -tuple $(\alpha(x_1), \dots, \alpha(x_k))$ belongs to $\alpha(P)$, the interpretation of predicate P under α . But $\alpha(P)$ is defined to include precisely those k -tuples of elements in A for which the corresponding predicate is assigned *true* by β (by construction of second half of [Theorem 9](#)). That is, $\alpha(P)$ has been defined to be $\{(\alpha(a_1), \dots, \alpha(a_k)) \mid \beta(\overline{PA_1 \dots A_k}) = \text{true}\}$. Because we have deduced that $(\alpha(x_1), \dots, \alpha(x_k))$ belongs to $\alpha(P)$, it must then have been the case that β assigns *true* to the proposition $\overline{PX_1 \dots X_k}$. Thus, in this case, β and β' indeed agree on the proposition in question.

- (b) If β' assigns *false* to $\overline{PX_1 \dots X_k}$, then the argument is similar to the one above, and is omitted for space.

We have shown that β and β' assign the same truth value to all propositions occurring in $NC(\Phi_2)$. Thus, β and β' must either both satisfy or both fail to satisfy $NC(\Phi_2)$. Since we have concluded above that $\beta' \models NC(\Phi_2)$, we can now conclude that $\beta \models NC(\Phi_2)$. But β was an arbitrary assignment that satisfies $NC(\Phi_1)$; we have shown that it also satisfies $NC(\Phi_2)$. Thus, any model of $NC(\Phi_1)$ is also a model of $NC(\Phi_2)$, and so $NC(\Phi_1) \models NC(\Phi_2)$, completing the proof. \square

We will also make use of the following result, which intuitively states that NC respects basic contradiction notions. While we will only use the result once in the following theorem, it is general enough to deserve being stated externally.

Lemma 17. *For any $GL^= KB \Psi$, $NC(\Psi) \cup NC(\neg\Psi)$ is unsatisfiable.*

Proof. Assume to the contrary. Then let Ψ be such a $GL^= KB$. Since it is satisfiable, it has a model; call it M . Since $M \models NC(\Psi) \cup NC(\neg\Psi)$, M is a model for both $NC(\Psi)$ and $NC(\neg\Psi)$ (a union of formulas is their conjunction, and thus an assignment which satisfies the union must satisfy all individual formulas in the conjunction). Let M' be a $GL^=$ interpretation obtained by using the construction described in the second part of [Theorem 9](#). By the result shown there, M' will satisfy a $GL^= KB \Phi$ if M satisfies $NC(\Phi)$. Because $M \models NC(\Psi)$, $M' \models \Psi$; and because $M \models NC(\neg\Psi)$, $M' \models \neg\Psi$. We now have $M' \models \Psi \cup \neg\Psi$, a contradiction⁶, showing that our assumption was incorrect, which completes the proof. \square

We are now ready to prove [Theorem 18](#), which we restate here for reference.

Theorem 18. *Let Φ be a $GL^= KB$ partitioned into Φ_1 and Φ_2 . Let Φ' be a $PL KB$ obtained as follows:*

$$\Phi' := NC(\Phi_1) \cup NC(\Phi_2) \tag{5}$$

Then, Φ and Φ' are satisfiability-equivalent ([Definition 5](#)). Equivalently, a conversion algorithm that applies [\(5\)](#) is satisfiability-equivalent.

⁶ Note that we do not need to show that this is a contradiction in $GL^=$, because we know it to be a contradiction in FOL, and $\Psi \cup \neg\Psi$ is an FOL formula, whereas M' is an FOL interpretation.

Proof. It is easy to show that $\text{SAT } \Phi$ implies $\text{SAT } \Phi'$. We show the other implication; that is, that $\text{SAT } \Phi'$ implies $\text{SAT } \Phi$. We will show the contrapositive. Assume $\text{UNSAT } \Phi$; that is, $\text{UNSAT } \Phi_1 \cup \Phi_2$. By Craig's Theorem (Proposition 15), there exist some $\text{GL}^\equiv \text{KB } \gamma$ such that:

- a. $\Phi_1 \models \gamma$;
- b. Φ_2 is inconsistent with γ , from which we can conclude
- c. $\Phi_2 \models \neg\gamma$; and
- d. The language of γ is the intersection of the languages of Φ_1 and Φ_2 .

Applying Lemma 16 to (a), we conclude (a') $\text{NC}(\Phi_1) \models \text{NC}(\gamma)$, and applying it to (c), we conclude (b') $\text{NC}(\Phi_2) \models \text{NC}(\neg\gamma)$. Assume, for contradiction, that $\text{SAT } \text{NC}(\Phi_1) \cup \text{NC}(\Phi_2)$. Then $\text{NC}(\Phi_1) \cup \text{NC}(\Phi_2)$ has a model, say M . Because M is a model of the union (conjunction) of formulas, it must also be a model for any subset of those formulas; thus, (c') $M \models \text{NC}(\Phi_1)$ and (d') $M \models \text{NC}(\Phi_2)$. From (a') and (c'), we can immediately conclude $M \models \text{NC}(\gamma)$, and from (b') and (d'), we can immediately conclude $M \models \text{NC}(\neg\gamma)$. Combining these results, we obtain $M \models \text{NC}(\gamma) \cup \text{NC}(\neg\gamma)$, so in particular $\text{SAT } \text{NC}(\gamma) \cup \text{NC}(\neg\gamma)$, which contradicts Lemma 17. Thus, our assumption must have been incorrect, which completes the proof. \square

The theoretical results achieved in this subsection allow us to create an intuitively significantly more efficient conversion method than the Naive Conversion, because we are applying the costly algorithms to smaller inputs. In addition to being a powerful result in the context of encoding GL^\equiv formulas, it is an intriguing theoretical result in its own right - one would not immediately expect an particular conversion algorithm to behave properly when applied to the fragments of its input.

5 Conclusion

In this section, we will describe related work, future work, and formulate a summary for this paper.

5.1 Related Work

A similar problem is solved for a different fragment of FOL in [4]. That work considers monadic FOL, which restricts predicates to being unary, but allows quantifiers. Our work applies to a fragment that is simultaneously more restrictive (since we disallow quantifiers), and less restrictive (we allow arbitrary arity of quantifiers; indeed, Section 3.3 provides an upper bound of the size of the output as a (exponential) function of the predicate arity).

5.2 Future Work

In our future work, we plan to build on the achieved results in several ways. The most obvious extension is to advance the partitioning to work recursively,

so that the input KB can be split into arbitrary large number of fragments, which are then encoded using NC. Because, as Section 3.3 has shown, the size of the output KB is directly dependent on size of the input $GL^=$ KB (and hence the running time of the final PL SAT solver is highly dependent on that size), employing NC on only small fragments can decrease the size of the resulting KB tremendously, thus achieving considerable saving in the SAT solver running time. The recursive fragmentation will be required to maintain the so-called running intersection property [6]; we plan to use a partitioning program already developed by one of us [9] for this purpose.

Noting that the "base case" conversion (NC in our case) is orthogonal to the partitioning mechanism, we also plan to improve the conversion algorithm to generate smaller-sized KBs. NC generates a significant amount of information, which represents a plethora of internal connections between the formulas of the original KB; indeed, it is such richness that made our final result (Theorem 18) possible. However, some of that information is extraneous for each particular case; specifically, some of the equivalence semantics formulas generated may not be required or even used in concrete cases. Any improvement in the underlying algorithm will result in propagated efficiency improvements in the complete application, because that underlying algorithm is invoked several times (two times in the described method, and much more in the recursive fragmentation extension proposed above, since NC will be used on every partition).

5.3 Summary

We have introduced $GL^=$, a decidable fragment of FOL. We have argued that $GL^=$ enjoys a unique position as being sufficiently expressive yet sufficiently simple, a position not shared by either general FOL (which has more expressive power, but suffers from inefficient deciding algorithms) or PL (which enjoys a plethora of very efficient SAT solving methods, but lack many constructs of FOL and thus can be hard to use for a particular application). In this context, we have formulated and thoroughly investigated a solution to general satisfiability problem by converting an arbitrary input knowledge base into a PL knowledge base, mandating that such conversion does not change satisfiability. We have presented several increasingly complex and increasingly efficient methods for such conversions, starting from Naive Conversion (which translated the input KB directly, ensuring both predicate and equality instance translation, and the equality semantics translation), to using divide-and-conquer partitioning paradigm in conjunction with Craig's Lemma in two different ways: both by putting the computed interpolants of one fragment into the resulting encoding, and by using Craig's Lemma to prove the ultimate contribution of this paper - Theorem 18. For each of these conversion methods, we have formulated the algorithm itself and proved that it preserves satisfiability. Specifically, Theorem 18 illustrates a very interesting theoretical result, which may prove to be useful beyond this paper - that an input KB can be divided into two fragments, which can then be encoded individually, and the union of the resulting encodings is satisfiability equivalent to the encoding of the entire KB. We believe that this theoretical

result, besides its obvious applicability for the purposes of encoding a $GL^=$ KB into PL, serves as an important example of the utility of the general divide-and-conquer paradigm.

References

1. Nance, M., Adam Vogel, E.A.: Reasoning about partially observed actions. In: AAAI (2006)
2. Baumgartner, P.: FDPLL – A First-Order Davis-Putnam-Logeman-Loveland Procedure. In: McAllester, D. (ed.) Automated Deduction - CADE-17. LNCS, vol. 1831, pp. 200–219. Springer, Heidelberg (2000)
3. Russel, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice-Hall, Englewood Cliffs (2003)
4. Ramachandran, D., Amir, E.: Compact propositional encodings of first-order theories. In: AAAI, pp. 340–345 (2005)
5. Amir, E.: Dividing and Conquering Logic. PhD thesis, Stanford University (2002)
6. Amir, E., McIlraith, S.: Partition-based logical reasoning for first-order and propositional theories. *Artificial Intelligence* 162(1-2), 49–88 (2005)
7. Craig, W.: Three uses of the herbrand-gentzen theorem in relating model theory and proof theory. *Journal of Symbolic Logic* 22(3), 269–285 (1957)
8. McMillan, K.L.: An interpolating theorem prover. *Theor. Comput. Sci.* 345(1), 101–121 (2005)
9. Amir, E.: Partitioning version 1.2. Technical report, Stanford University Software description (2002), available at <http://reason.cs.uiuc.edu/eyal/decomp/README>

Tailoring Solver-Independent Constraint Models: A Case Study with ESSENCE' and MINION

Ian P. Gent, Ian Miguel, and Andrea Rendl

School of Computer Science, University of St Andrews, UK
{ipg, iamm, andrea}@cs.st-andrews.ac.uk

Abstract. In order to apply constraint programming to a particular domain, the problem must first be *modelled* as a constraint satisfaction problem. There are typically many alternative models of a given problem, and formulating an effective model requires a great deal of expertise. To reduce this bottleneck, the ESSENCE language allows the specification of a problem *abstractly*, i.e. without making modelling decisions. This specification is refined automatically by the CONJURE system to a solver-independent constraint modelling language ESSENCE'. However, there is still significant work involved in translating an ESSENCE' model for use with a particular constraint solver. This paper discusses this 'tailoring' process with reference to the constraint solver MINION.

1 Introduction

Constraint programming is a successful technology for tackling a wide variety of combinatorial problems. To use constraint technology to solve a problem, the problem must first be described in terms of a *constraint model* suitable for input to a *constraint solver*, which then searches for solutions automatically. The process of formulating an *effective* constraint model (i.e. for which the intended constraint solver is able to find solutions efficiently) is notoriously difficult and is one of the major bottlenecks preventing the wider use of constraint solving.

Hence, automating constraint modelling is highly desirable. In one approach the user provides an abstract problem specification in which detailed modelling decisions have not yet been taken. This specification is then *refined* automatically into a constraint model. The ESSENCE abstract constraint specification language [2] and CONJURE automated refinement system [4] embody this approach. The constraint models produced by CONJURE are in the language ESSENCE', which has a level of abstraction supported by existing constraint solvers.

ESSENCE' is, however, a solver-independent constraint language; ESSENCE' models must undergo a further translation step to produce input suitable for any particular constraint solver. The difficulty of this task depends on the facilities offered by the intended solver. Moreover, as shown by Prosser and Selenksy [10], individual constraint solvers have differing strengths and weaknesses. Therefore, an ESSENCE' model must be *tailored* to an individual solver to maximise efficiency. This paper considers the tailoring process, with particular reference to the constraint solver MINION [6]. This is a particular challenge, since MINION has been deliberately pared down to increase solving efficiency.

2 Background

We begin by providing the necessary background in constraint modelling and solving before describing the ESSENCE' language and MINION constraint solver.

2.1 Constraint Satisfaction Problems and the Modelling Bottleneck

The finite-domain *constraint satisfaction problem* (CSP) consists of: a finite set of decision variables, \mathcal{X} ; for each variable $x \in \mathcal{X}$, a finite set $\mathcal{D}(x)$ of values (its domain); and a finite set \mathcal{C} of constraints on the variables, where each constraint $c \in \mathcal{C}$ is defined over a subset of $\{x_i, \dots, x_j\}$ of \mathcal{X} (its *scope*, denoted $\text{scope}(c)$) by a subset of the Cartesian product $\mathcal{D}(x_i) \times \dots \times \mathcal{D}(x_j)$ giving the set of allowed combinations of values. A constraint can be specified extensionally, by listing these tuples, or intensionally as an expression with an associated *propagation* algorithm that is executed by the constraint solver to determine satisfaction/violation. A solution to a CSP assigns values to all variables such that all constraints are satisfied.

A constraint model maps the features of a combinatorial problem onto the features of a CSP. The CSP is input to a constraint solver, which searches for a solution (or solutions). The constraint model is then used to map the solution(s) back onto the original problem. Constraint languages and solvers commonly provide a rich library of constraints from which to choose. Typically, therefore, many models are possible for a given problem. This choice is both complex and important: it can mean the difference between the problem being solved quickly and not being solvable in a practical amount of time. Hence, constraint modelling is difficult and requires a great deal of expertise.

The constraint specification language ESSENCE [2] addresses this modelling bottleneck. It enables the specification of a combinatorial problem *abstractly*, without making constraint modelling decisions. This is achieved by supporting decision variables whose domain elements are the combinatorial objects (e.g. functions, relations) that a combinatorial problem commonly requires us to find, but which are not directly supported by existing constraint solvers.

To illustrate, consider Langford's problem (CSPlib problem 24), which is to arrange n sets of positive integers $1..k$ into a sequence such that, following the first occurrence of an integer i , each subsequent occurrence of i appears $i + 1$ indices later than the last. For example, if k is 4 and n is 2, then a solution is 41312432. The problem may be viewed as requiring us to find a bijection from the values to their positions in the sequence. This can be stated directly in ESSENCE, as Figure 1 shows. For simplicity, we set $n = 2$, hence the sequence

given	$k : \text{int}(1 \dots)$
find	$\text{positions} : \text{int}(1..2 * k) \rightarrow (\text{bijective}) \text{int}(1..2 * k)$
such that	forall $i : \text{int}(1..k) . \text{positions}(i + k) - \text{positions}(i) = i + 1$

Fig. 1. An ESSENCE specification of a simplified version of Langford's problem

has length $2k$. We use $2k$ values to represent the elements of the sequence. The first occurrence of integer i is represented by i itself, and the second occurrence by $i+k$. The key feature of this specification is that it contains just one decision variable, *positions*, whose domain is the set of possible bijections from $1..2k$ onto $1..2k$. The single constraint ensures that the pairs of elements are the requisite number of indices apart. The bijection in this specification can be modelled in a variety of ways [9], but ESSENCE does not force the user to make this decision.

2.2 The ESSENCE' Solver-Independent Modelling Language

Given an ESSENCE specification, constraint modelling consists in encoding the abstract decision variables and the constraints on them as constrained collections of CSP variables. The CONJURE [4] automated refinement system performs this step automatically, producing a constraint model in a subset of ESSENCE called ESSENCE'. ESSENCE' is a solver-independent constraint modelling language with a level of abstraction that is supported by existing constraint solvers. ESSENCE' is intended to be very nearly an object-level language. However, as we demonstrate in this paper, there remain issues in translating to a specific language that must be resolved carefully to avoid affecting performance adversely.

Decision variables are specified individually or as elements of multi-dimensional matrices. Their domains may contain integers or Booleans only, in contrast with the abstract decision variables offered by ESSENCE. The available constraint library includes basic building blocks, such as the ability to specify constraints extensionally, and arithmetic and logical operators, which can be nested to form arbitrarily complex constraint expressions. It also includes commonly-used intensional constraints such as the all-different constraint [11], which constrains a matrix of variables to take distinct values, and the lexicographic ordering constraint [3], which can be used to deal with symmetry in constraint models. Existential and Universal quantifiers are also supported and may be nested.

To illustrate, Figure 2 shows an ESSENCE' model of the simplified Langford's problem specified in Figure 1. The refinement here is simple: the function is modelled using a matrix of decision variables indexed by the values in the sequence. The domain of each decision variable is the set of possible positions. Note the use of the all-different constraint to ensure that the bijective property holds.

1	given k : int(1..)
2	find positions : matrix indexed by [int(1..2*k)] of int(1..2*k)
3	such that
4	forall i : int(1..k) .
5	positions[i+k] = positions[i] + i+1,
6	allDifferent(positions)

Fig. 2. ESSENCE' model of a simplified version of Langford's problem

2.3 The MINION Constraint Solver

The MINION [6] constraint solver is designed to promote solving efficiency. Consequently, its input language is relatively restricted, but it does offer the option to tune various low-level features that are not commonly accessible.

Like ESSENCE', MINION supports decision variables with integer domains, which can be collected in multi-dimensional matrices. It also provides fine control over domain representation. MINION supports four individually-optimised integer domain types: 0/1 domains, commonly used for logical expressions and counting; *bounds* domains, which maintain only the lower and upper bound of the domain; *sparse* domains, where domain values need not be adjacent, e.g. {2, 7, 11}; and *discrete* domains, which are defined initially by lower and upper bounds but which support the removal of values from between the bounds.

Table 1. MINION constraints: x and r refer to decision variables, v to a decision variable vector, and c refers to a constant. Lists of decision variables $[x_1, \dots, x_n]$ may be replaced by vectors and rows or columns of matrices.

MINION Constraints	Meaning
<code>sumleq([x1,x2,...,xn], r)</code>	$x_1 + x_2 + \dots + x_n \leq r$
<code>sumgeq([x1,x2,...,xn], r)</code>	$x_1 + x_2 + \dots + x_n \geq r$
<code>weightedsumleq([x1,...,xn],[c1,...,cn], r)</code>	$x_1 * c_1 + \dots + x_n * c_n \leq r$
<code>weightedsumgeq([x1,...,xn],[c1,...,cn], r)</code>	$x_1 * c_1 + \dots + x_n * c_n \geq r$
<code>product(x1, x2, r)</code>	$x_1 * x_2 = r$
<code>eq(x1,x2)</code>	$x_1 = x_2$
<code>diseq(x1,x2)</code>	$x_1 \neq x_2$
<code>ineq(x1,x2,c)</code>	$x_1 \leq x_2 + c$
<code>max([x1,...,xn],r)</code>	$max(x_1, \dots, x_n) = r$
<code>min([x1,...,xn],r)</code>	$min(x_1, \dots, x_n) = r$
<code>element(v,i,r)</code>	$v[i] = r$
<code>alldiff(x1,...,xn)</code>	$x_1 \neq x_2 \neq \dots \neq x_n$
<code>reify(constraint,r)</code>	if(<i>constraint</i>) then $r = 1$ else $r = 0$
<code>table(matrix, tuple)</code>	an extensional constraint

1	0	.
2	8	.
3	1 8 8	8
4	0	sumgeq([x0, 2], x4)
5	0	sumleq([x0, 2], x4)
.	.	10
.	.	sumgeq([x1, 3], x5)
6	1	11
.	.	sumleq([x1, 3], x5)
.	.	12
7	[x0,x1,x2,x3,x4, x5, x6, x7],	sumgeq([x2, 4], x6)
.	.	13
.	.	sumleq([x2, 4], x6)
.	.	14
.	.	sumleq([x3, 5], x7)
.	.	15
.	.	sumleq([x3, 5], x7)
.	.	16
.	.	alldiff(v0)

Fig. 3. Partial MINION instance of the simplified Langford’s problem ($k = 4$)

MINION offers a broadly similar set of constraints (summarised in Table II) compared with ESSENCE', but again provides controls as to how they are implemented, as will be explained below. A key difference is that MINION supports neither quantification nor nested constraint expressions. It also allows the statement of individual *instances* only, rather than parameterised problem classes.

To illustrate, Figure 3 presents part of the MINION input file for the instance of Langford’s problem when $k = 4$. The MINION manual [8] contains full details of the input format. Briefly, in a model with n variables, each variable is identified by ‘ x_i ’, for i in $0..(n - 1)$. MINION insists that variables of each type described above are declared in turn. In the example there are 8 bounds variables with lower bound 1 and upper bound 8 (lines 2-3). These variables are collected in the vector v_0 (lines 6-7). The single universally-quantified constraint from the ESSENCE’ model is expanded into a set of individual sum constraints (lines 8-15). Note that MINION requires that each equality constraint is decomposed into a pair of sum constraints. The all-different constraint is added directly (line 16).

3 Tailoring ESSENCE’ to MINION: Overview

Since MINION expects individual instances, the input to the tailoring process is a pair: an ESSENCE’ model, and a set of values for the parameters of the model sufficient to determine a particular instance. The first (straightforward) step in the tailoring process is therefore to parse the ESSENCE’ model and, for each parameter, substitute its given value for each of its occurrences in the model. Following substitution, a simplification step is performed to evaluate expressions now composed entirely of constants. These pre-processing steps are independent of the target solver.

In translating ESSENCE’ to MINION, we make use of a `MinionModel` structure, a four-tuple $\langle V, A, C, M \rangle$ where: V is the set of MINION decision variables; A is the set of MINION matrices, whose elements are drawn from V ; C is the set of MINION constraints; and M is a bijection between the elements of V and the original ESSENCE’ variables. The function M is important both for mapping solutions produced by MINION back onto the ESSENCE’ model and to avoid the repeated introduction of MINION variables for a single ESSENCE’ variable that occurs in several constraints.

We define a translation function τ as follows:

$$\tau : \langle \mu, e \rangle \rightarrow \mu'$$

where μ is an instance of a `MinionModel` and e is an ESSENCE’ expression. This pair is mapped to a new `MinionModel` instance μ' . The effect of applying τ is to increase monotonically the four elements of the `MinionModel`. Beginning with an empty `MinionModel`, tailoring of an ESSENCE’ model proceeds through the constraint-wise application of τ , incrementally constructing the final model.

Since ESSENCE’ is the richer language, typically each ESSENCE’ constraint corresponds to a set of MINION constraints. Variables are introduced into the `MinionModel` both to correspond to the variables in the original ESSENCE’ model (in which case the correspondence is recorded in the mapping M) and to support the decomposition of an ESSENCE’ expression. Decomposition is necessary to deal both with nested expressions and with the quantifiers and constraints not directly supported by MINION, as is described in the following sections.

4 Arithmetic Constraints

We begin by discussing the translation of arithmetic expressions. As noted, MINION provides the small set of constraints given in Table 1. ESSENCE' constraints of exactly this form require no translation. For the remainder, simple *reformulation* steps are performed. Note, for example, that MINION does not provide a division operator. Hence, division in ESSENCE' is translated by rewriting division to multiplication. In general an ESSENCE' arithmetic expression is translated via combinations of the primitive constraints in Table 1, sometimes connected by introducing auxiliary variables. A very simple example can be seen in the MINION instance of Langford's problem (Figure 3): to constrain a sum of variables and constants to be equal to a variable or a constant, a pair of `sumgeq` and `sumleq` constraints is used (e.g. lines 8-9). Equality of weighted sums is treated similarly.

Commonly, ESSENCE' arithmetic constraints are translated by *flattening* nested expressions. That is, an ESSENCE' expression is decomposed into sub-expressions for which MINION provides a corresponding constraint. All MINION constraints used for this purpose are expressed in terms of equality or inequality relations, as per the examples in Table 1. Hence, an auxiliary variable is constrained to be equal to each sub-expression. Constraints among the auxiliary variables are added as necessary to create a set of MINION constraints equivalent (i.e. collectively allowing the same set of assignments) to the original ESSENCE' constraint.

To illustrate, consider an ESSENCE' constraint that constrains the sum of n variables to be not equal to some variable r . MINION provides a binary disequality constraint, but no direct way to express a disequality on a sum. Hence, it is natural to introduce an auxiliary variable constrained, in the same way as above, to be equal to the sum of the n variables *and* to be not equal to r :

ESSENCE'	MINION
$x_1 + \dots + x_n \neq r$	<code>sumleq([x1,x2,...,xn], a)</code> <code>sumgeq([x1,x2,...,xn], a)</code> <code>diseq(a, r)</code>

Equality and disequality of weighted sums are treated similarly.

When an auxiliary variable is introduced it must be given an appropriate domain. In doing so, we can exploit our knowledge of the MINION solver. Propagation of each of the constraints in Table 1 affects only the bounds of the variable r . Hence, it is sufficient to use an efficient bounds domain (see Section 2.3) for each auxiliary variable introduced in translating an arithmetic expression. The lower and upper bound of the domain of an auxiliary variable is determined by examining the lower and upper bounds of the expression to which it is constrained to be equal. Hence, in the current example the domain of a ranges from the sum of the lower bounds of x_1, \dots, x_n to the sum of their upper bounds.

Our general method of translating nested ESSENCE' expressions proceeds from the parse tree as follows:

1. Consider the nested ESSENCE' expression a tree with variables and constants as leaves, operators as nodes and the tree branched according to the operator precedences where the operator with lowest precedence is root.
2. Take an operator node op with highest depth that connects leaves $l_1..l_n$. Generate the corresponding MINION constraint(s) for $op(l_1, \dots, l_n) = v$ where v is a auxiliary variable. Add the generated constraints to C and v to V .
3. If the tree contains at least another leaf, replace node op by leaf v and go to 2., otherwise stop.

To minimise the generated overhead of variables and constraints during flattening, we can apply simple and effective rules: directly match MINION primitives to the expression tree structure, such as (weighted) sums or products. With a subtree matching an iterated sum structure with n leaves, this strategy reduces the number of auxiliary variables from $n - 1$ (or n for the whole tree) to 1.

5 Logical Constraints

Boolean ESSENCE' expressions are translated using 0/1 variables and arithmetic constraints. Table 2 summarises how the common logical connectives are translated for MINION. As in the arithmetic case, flattening is required for nested logical expressions. This operates in broadly the same way as for arithmetic, but differs in that it requires *reification*. Reification can be viewed as a 'meta' constraint in that it equates the satisfaction of some constraint c with a Boolean variable. MINION provides reification using 0/1 variables and the constraint: `reify(c, x1)`, meaning $x1$ is assigned 1 if and only if c is satisfied.

To illustrate, consider the ESSENCE' expression $(x1 = x2) \Rightarrow (x3 = 0)$. MINION does not provide a constraint of this form, so flattening is required. To do so, we decompose the implication into left- and right-hand sides and reify them into two auxiliary 0/1 variables: `reify(eq(x1, x2), a1)`, `reify(eq(x3, 0), a2)`. The implication can now be stated using an inequality: `ineq(a1, a2, 0)`.

Not all constraints in MINION can be reified, hence care must be taken to employ only reifiable constraints for a subexpression if it will be reified. In order

Table 2. Basic logical connectives and their equivalents using MINION 0/1 variables. 'n' is a unary operator on a MINION 0/1 variable x that returns $1-x$. Clearly, the sum constraint is only necessary to express conjunction arising in some nested sub-expression. Otherwise e_1 and e_2 can simply be imposed separately, since a solution requires all constraints to be satisfied. '`ineq(x1, x2, c)`' is interpreted $x_1 \leq x_2 + c$. If the e_i are nested expressions, flattening is required.

Logical Expression	MINION Constraint
$\neg e$	<code>nx</code>
$e_1 \wedge e_2$	<code>sumgeq([x1, x2], 2)</code>
$e_1 \vee e_2$	<code>sumgeq([x1, x2], 1)</code>
$e_1 \Rightarrow e_2$	<code>ineq(x1, x2, 0)</code>
$e_1 \Leftrightarrow e_2$	<code>eq(x1, x2)</code>

to determine if a subexpression will be reified or not, we need to have information about its *context*. This is why we introduce a *reification flag*, that indicates when *true* that the currently translated expression will to be reified. The reification flag is initially false, retains its state as we traverse certain constraints (e.g. a conjunction) but becomes true when we traverse other constraints (e.g. a disjunction).

An interesting case of reformulation occurs with negation: ESSENCE' supports negation of logical expressions while MINION only allows the negation of single variables (see Table 2). This is why negation is applied to expressions before translation. Negated relational expressions are reformulated by applying the corresponding complementary operator, for example $\neg(x = y)$ is reformulated to $x \neq y$. Negated Boolean expressions are reformulated by applying Boolean axioms: $\neg(x \wedge y)$ results in $\neg x \vee \neg y$. In the last case the negation operator is passed down a level in the expression tree, eventually being applied to a variable or relational expression. Hence, the reformulation process halts, even though the worst case may take exponential time in the depth of the expression tree.

Generally, flattening of logical expressions proceeds directly from the parse tree, as described in Section 4, but making use of reification rather than arithmetic constraints for decomposition. However, care must be taken in the presence of universal and/or existential quantification. Quantified expressions in ESSENCE' have the form $q \ i_1, \dots, i_n \in D. e(i_1, \dots, i_n)$ where $q \in \{\forall, \exists\}$ is a quantifier, i_1, \dots, i_n are binding variables that range over the finite integer domain D and $e(i_1, \dots, i_n)$ is an arbitrary relational expression involving $i_1..i_n$. Translation of quantified expressions is further complicated by the fact that nesting is allowed, i.e. e may also contain quantified expressions. In what follows, we describe a general, effective approach to translating arbitrarily-nested quantified ESSENCE' expressions into MINION.

5.1 Singly-Quantified Expressions

We first define a basic approach for translating quantified expressions. Universal quantification can be treated as a conjunction, existential quantification as a disjunction of expressions. Consider the example $\forall_{i \in [1..5]}. (m[i] \neq i)$. It corresponds to the conjunction $(m[1] \neq 1) \wedge \dots \wedge (m[5] \neq 5)$, and can be translated to 5 separate constraints. Now consider $\exists_{i \in [1..5]}. (m[i] \neq i)$ that corresponds to a disjunction $(m[1] \neq 1) \vee \dots \vee (m[5] \neq 5)$. In this case we need to apply reification. Imposing `reify(diseq(m[1], 1), x1)` gives us a reified variable `x1` that is set to 1 if `diseq(m[1], 1)` holds and 0 if not. So the disjunction is satisfied, if at least one reified variable equals to 1. We can enforce that by imposing another constraint, stating that the maximum of all reified variables has to equal 1: `sumgeq([x1, x2, x3, x4, x5], 1)`. Conjunction can be translated by insisting that the sum of the k reified variables equals k . Thus the translation of disjunction introduces n auxiliary variables and $n + 1$ reification constraints with $n = |D|$ where D is the domain of binding variable i . Hence, in the general case with m binding variables over domain D , we get n^m additional variables, $n^m + 1$ reification constraints, and n^m expressions to translate.

Table 3. Translation of quantifications; $m : e \rightarrow c$ returns the MINION constraint corresponding to ESSENCE' expression e and $k = |D|^n$

ESSENCE' expression	MINION constraints	Auxiliary variables
$\exists i_1..i_n \in D.(e)$	$\{ \text{reify}(m(e(i_1,..i_n)), \text{x1}) \mid i_1..i_n \in D \}$ $\text{sumgeq}([x1, ..xk], 1)$	$\text{x1}..xk$
$\forall i_1..i_n \in D.(e)$ $\text{reify} = \text{false}$	$\{ m(e(i_1,..i_n)) \mid i_1..i_n \in D \}$	none
$\forall i_1..i_n \in D.(e)$ $\text{reify} = \text{true}$	$\{ \text{reify}(m(e(i_1,..i_n)), \text{x1}) \mid i_1..i_n \in D \}$ $\text{sumgeq}([x1, ..xk], k)$ $\text{sumleq}([x1, ..xk], k)$	$\text{x1}..xk$

5.2 Nested Quantification

While existential quantification, as a form of disjunction, can only be translated using reification, universal quantification is a form of conjunction and sometimes can be translated without reification. We saw above that the expression $\forall_{i \in [1..5]}.(m[i] \neq i)$ can be translated without reification. Generally, a translation without reification is to be preferred as being simpler and allowing propagation more easily. Unfortunately, reification is sometimes necessary where universally quantified constraints are nested. For example, if x is some constrained variable, the expression $x = 1 \Rightarrow \forall_{i \in [1..5]}.(m[i] \neq i)$ will require the universal constraint to be reified to a variable r and then the constraint $x = 1 \Rightarrow r = 1$ posted. This shows another application of the reification flag.

Generally, we translate quantifiers by inserting values for their binding variables and translate the resulting expressions according to the imposed quantifiers. We apply values for binding variables recursively, starting with the outmost quantifier, and then stepwise increase the values, beginning with the first variable of the innermost quantifier. If a variable has reached its upper bound, it is reset to its lower bound and the next variable's value is increased. When the last binding variable of the outmost quantifier has reached its upper bound, we stop. During this value-insertion process we build the resulting constraints from the inside out: Depending on the quantifier and the reification flag, we generate a set of constraints. Consider the quantification $\exists_{i \in [0..5]}\forall_{j \in [1..3]}.m[j] \neq i$, that corresponds to a disjunction of conjunctions of $m[j] \neq i$, as shown below.

$$\begin{aligned} \exists_{i \in [0..5]}\forall_{j \in [1..3]}.(m[j] \neq i) = & \\ & \bigvee m[1] \neq 0 \wedge m[2] \neq 0 \wedge m[3] \neq 0 \\ & \bigvee m[1] \neq 1 \wedge m[2] \neq 1 \wedge m[3] \neq 1 \\ & \bigvee \dots \\ & \bigvee m[1] \neq 5 \wedge m[2] \neq 5 \wedge m[3] \neq 5 \end{aligned}$$

The conjoined expressions $m[j] \neq i$ are in a disjunctive context because of the outermost existential, so we set the reification flag to be *true*. First we insert lower bounds of variables i, j and then increase j 's value until we reach its upper bound, getting the set of expressions $m[1] \neq 0, m[2] \neq 0, m[3] \neq 0$. We impose the

innermost quantifier, \forall , giving us a conjunction to translate: $m[1] \neq 0 \wedge m[2] \neq 0 \wedge m[3] \neq 0$. With the reification flag being *true*, we have to reify the expressions and get 3 reification constraints `reify(m[1] \neq 0, x1) ... reify(m[3] \neq 0, x3)` and a sum constraint `sumgeq([x1, x2, x3], 3)`, introducing 3 auxiliary variables `x1`, `x2`, `x3`. Since i can take 6 different values, we have 6 conjunctions to translate, resulting to 18 reification constraints and 6 sum constraints. We now impose the \exists quantifier on our set of conjunctions. Each conjunction is represented by a sum constraint that we reify: `reify(sumgeq([x1, x2, x3], 3), x00)`, .. `reify(sumgeq([x16, x17, x18], 3), x05)` and express disjunction by the sum `sumgeq([x00, x01, x02, x03, x04, x05], 1)` over the auxiliary variables `x00` .. `x05`, according to Table 3.

5.3 Treating Special Cases

These rules are all applied in the basic implementation. They allow us to translate expression “on the fly”, since it can be immediately determined which rule to apply and the translation does not depend on future events (the translation is causal). Consequently, we only employ a small amount of memory during translation. However, there are cases where we translate quantified expressions that are later shown to be redundant. Consider the example $\exists_{i \in [1..n]}. e \vee i = n$ where e is an arbitrary expression. Since n is in the range of i , the expression will evaluate to *true*. The existential quantification, corresponding to disjunction, also becomes *true*. Please note, that this case may only be spotted at instance level, since the domain of a binding variable may depend on a parameter value. To detect cases where quantified expressions are always satisfied or violated requires significant effort. We will investigate this procedure in future, but are aware that the overhead might outweigh the benefit gained.

6 Global Constraints

Global Constraints [13] represent general problem patterns, e.g. that every element of a datastructure has a distinct value is captured by the global constraint *alldifferent* [11]. Constraint solvers usually provide efficient propagators for global constraints. ESSENCE' provides a range of global constraints which can be directly mapped to the corresponding MINION global constraint, or an equivalent set of constraints added if no exact equivalent is available.

MINION provides different versions of some constraints, giving us choices we have to make. In most cases the differences arise from the type of propagation, through the use ‘watched literals’, a recently-introduced method for writing constraint propagators [7]. Examples are `sum` and `element`. Watched literal-based constraints can reduce the search time drastically [7]. However, classical propagators can still be more effective in some cases. We generally choose watched literal-based constraints, but give the user the possibility to force unwatched constraints to be applied by setting a flag. The current version of MINION does not support reification of watched literal-based constraints, so unwatched constraints are always chosen in a context where the reification flag is true.

MINION provides matrix indexing by decision variables using the *element* constraint, `element(matrix, index, elem)`, stating that the element at position `index` of (one- or multidimensional) matrix `matrix` equals to (the assignment of) `elem` [12]. We use this to translate matrix indexing in ESSENCE'. For example, the simple expression $m[1, x] = n$ can be expressed by `element(row(m, 1), x, n)`. If dynamically indexed matrices occur in nested expressions, we need to introduce an auxiliary variable to represent the indexed matrix element. Furthermore, we are able to nest dynamic indexing and express a $k - 1$ times nested indexing with k element constraints. For instance, $m[1, v[x]] = n$, which is nested once, can be reformulated to 2 element constraints, `element(row(m, 1), tmp, n)` and `element(v, x, tmp)` by introducing the auxiliary variable `tmp`.

Matrices that are entirely indexed by decision variables need to be flattened. We illustrate this case by considering the Mutually Orthogonal Latin Squares (MOLS) problem [15]: a latin square is an $m \times m$ matrix of elements $1..m$ where each row and column has distinct elements. Two latin squares A and B are mutually orthogonal, if each pair of elements $(A(i, j), B(i, j))$ occurs exactly once, as illustrated in the example below. We model this problem by introducing two auxiliary matrices X and Y , holding the row and column indices of A and B 's elements such that $A[X[i, j], Y[i, j]] = i$ and $B[X[i, j], Y[i, j]] = j$. If such matrices X, Y exist, then A and B are mutually orthogonal.

`element` is restricted to one index parameter, hence translating an expression $A[x, y] = c$ requires flattening matrix A to a vector A' where $A'[i * r + j] = A[i, j]$ and r corresponds to the amount of rows in matrix A . Hence we obtain the element constraint `element(A', i, c)` with an adjusted index $i = x * r + y$. Indexing matrices with decision variables without the need to flatten them by hand is essential, because it allows to express further constraints that are more easily expressed over matrix structures: consider the MOLS problem again, where we need to impose *alldifferent* on all rows and columns of matrices A and B . This can be more easily expressed over a matrix structure than a flattened vector structure.

7 Variable Translation

As mentioned in Section 2.3, MINION has different types of integer decision variables: bounds- and discrete-domain variables. Discrete domain variables allow deletion of values inside the bounds during search, which can be very effective, for instance in combination with the *alldifferent* constraint. Where constraints such as this are used, use of bounds-domain variables in MINION risks run-time errors on attempted value removals. In ESSENCE' there is no such distinction of bound variables, and so we generally translate decision variables to discrete-domain variables. We allow the user to select bounds-domain variables for translation by a flag if they are confident that discrete domains are unnecessary.

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \\ 2 & 3 & 1 \end{pmatrix} \quad (A(i, j), B(i, j)) = \begin{pmatrix} (1, 1) & (2, 2) & (3, 3) \\ (2, 3) & (3, 1) & (1, 2) \\ (3, 2) & (1, 3) & (2, 1) \end{pmatrix}$$

When a variable x is assigned to another variable y , such as in the ESSENCE' expression $y = x$, we can 'reuse' x by substituting x for any occurrence of y . This procedure can also be applied for (parts of) matrices, since in MINION decision variables can occur in several different matrices. Consider the example $\forall_{i \in 1..10}. v[i] = u[i+1]$, where we assign every second element of vector u to vector v . Here we construct vector v out of the corresponding elements of u .

8 Experimental Results

We have implemented a translator from ESSENCE' to MINION in Java 1.5.0_10. In this section we report results using MINION on translated instances. Our translator is under active development and can be found at minion.sourceforge.net.

In this section we present some problems that we have specified in ESSENCE' and tailored to a set of MINION instances. We compare these instances with optimised benchmarks produced by problem-specific instance generators provided by MINION. These generators have been implemented by experts in both constraint modelling and MINION. Hence we are comparing automatically tailored instances with human expertise. We performed our experiments on a 3GHZ Pentium 4 with 2GB RAM using MINION version 0.4.1 compiled with g++ 4.0.2 under Linux. The translator takes a negligible amount of time for translation, hence we do not include its runtime and focus on how competitive the resulting instances are.

8.1 The Balanced Incomplete Block Design (BIBD)

The BIBD (CSPlib problem 28) is defined by a 5-tuple of positive integers $\langle v, b, r, k, \lambda \rangle$: assign v objects to b blocks such that each block contains k different objects and exactly r objects occur in each block and every two distinct objects occur in exactly λ blocks. A typical problem model consists of a $0/1$ matrix m with b columns (blocks) and v rows (objects), where an element m_{ij} is assigned 1 if block i contains object j . Each row is constrained to the sum r and each column to the sum k . The scalar product of each two rows corresponds to λ . We order rows and columns lexicographically to break symmetry partially.

As summarised in Table 4, we compared generated MINION instances with BIBD instances generated by the hand coded BIBD-instance generator provided by MINION, with constraints using watched literals and without. Our translator produced almost identical instances to those produced by the instance-generator,

Table 4. Results for solving the BIBD problem

b, v, r, k, λ	Unwatched				Watched			
	Time(sec)		Nodes		Time(sec)		Nodes	
	Gen.	Trans.	Gen.	Trans.	Gen.	Trans.	Gen.	Trans.
140,7,60,3,20	0.51	0.51	17235	17235	0.67	0.7	17235	17235
210,7,90,3,30	2.08	2.04	67040	67040	3.03	3.07	67040	67040
280,7,120,3,40	6.73	6.51	182970	182970	9.45	9.34	182970	182970
315,7,135,4,45	10.73	10.67	278310	278310	15.14	15.32	278310	278310

Table 5. Results for finding a solution to the n-Queens problem

n	Time(sec)			Nodes		
	Generator	Translator		Generator	Translator	
	discrete var.	discrete var.	bounds var.	discrete var.	discrete var.	bounds var.
12	< 0.01	< 0.01	< 0.01	60	59	1,840
15	< 0.01	< 0.01	0.02	249	248	12,687
17	0.01	0.02	0.13	1,187	1,186	62,382
19	< 0.01	0.01	0.07	583	582	34,595
20	0.48	0.65	5.63	37331	37,330	2,857,524
22	3.84	5.19	55.74	269,370	269,369	28,458,527
24	1.01	1.34	15.19	63,791	63,790	7,528,769
25	0.12	0.16	2.08	7,272	7,271	943,172
26	0.96	1.27	16.97	55,592	55,591	8,057,222
27	1.16	1.56	20.47	67,231	67,230	9,723,687
29	3.94	5.09	72.69	212,276	212,275	35,867,550
30	141.24	193.57	>45min	7,472,996	7,472,995	1,379,220,754

performing almost exactly the same in means of time and identically in amount of search nodes used. This demonstrates the effectiveness of the tailoring process for the BIBD problem.

8.2 The n-Queens Problem

The n queens problem is to place n queens on a $n \times n$ chess board without attacking each other. In the problem model, the queens are represented by a vector v of length n where the element v_i corresponds to the column of the queen placed in row i . No queen may be placed on the same diagonal as another, which is specified using two auxiliary vectors of same length n . Each element of the auxiliary vectors has a distinct domain, which cannot be encoded in ESSENCE' and therefore has to be restricted by additional constraints. We produce instances with both bound-domain and discrete-domain variables and compare them to instances generated by an n-queens instance-generator for the same model provided with the MINION distribution. This generator creates instances with discrete-domain variables and distinct bounds for the auxiliary variables. Results are given in Table 5. We see that the run times are notably slower than the hand-written generator, up to just under 40% in the largest instance. Nodes searched is always exactly one less. In general terms, these results show that we produce good models, but unsurprisingly there can be scope for further optimisation if writing a specialised generator. We can also observe the drawback of bound-domain variables with this problem.

8.3 The Quasigroup Problem

An m order quasigroup is an $m \times m$ multiplication table of integers $1..m$, where each element occurs exactly once in each row and column and certain multiplication axioms hold. The quasigroup problem (CSPLib problem 3) is concerned with the existence of such a group of order m . We compared instances of the generator and translator with and without a special variable ordering. Variable orderings have been added by hand after the translation process since ESSENCE' has, at the time of writing, no facilities to specify variable orderings. Both instances

Table 6. Results for solving the QuasiGroup7 existence problem of order m

m	with Ordering				without Ordering			
	Time(sec)		Nodes		Time(sec)		Nodes	
	Trans.	Gen.	Trans.	Gen.	Trans.	Gen.	Trans.	Gen.
7	< 0.01	< 0.01	756	844	0.03	0.03	3,275	3,272
8	0.1	0.1	11,949	12,450	1.94	1.89	171,203	169,078
9	< 0.01	< 0.01	238	233	5.15	5.23	458,062	454,512
10	249.6	250.02	30,549,274	31,383,717	>1h	>1h	-	-

apply the same specified ordering. As demonstrated by the results in Table 6, an efficient variable ordering is crucial for solving the problem efficiently. Such an ordering can easily be added by the user in the MINION instance. Our tailored instances have shown a slightly better performance with variable orderings and are therefore highly competitive to the generated ones.

8.4 Summary

Our experimental results are not extensive, but show that instances tailored by our translator tend to perform well in comparison with those produced by instance generators implemented by modelling experts. Developing a generator takes significantly more time and knowledge about MINION than simply expressing a problem in ESSENCE'. We see in one case that the hand-coded generator runs faster, showing that (as expected) we cannot always attain optimal encodings automatically. Finally, our quasigroup results raise an important issue. Specifying good heuristics is often not considered to be a part of modelling but is well known to be essential to success. The more successful tools such as CONJURE and our translator become, the more important it will be to specify heuristics during this process, either manually or automatically.

There are a number of outstanding issues. The most important of these is that the only global constraints supported are alldifferent and element. To correct this is trivial for global constraints supported by Minion (e.g. table) but requires implementing an encoding of constraints which are not supported directly (e.g. global cardinality).

9 Conclusion

This paper has discussed the issues arising in tailoring models in a solver-independent constraint language, ESSENCE', to the constraint solver MINION. This process may be compared with that of translating OPL to Ilog Solver, which formed part of a commercial product from Ilog. However, to the best of our knowledge, details of the OPL to Solver translation are unpublished. Furthermore OPL lacks, for example, existential quantification which, when nested, significantly complicates the translation process, as we have seen. Charnley *et al* [11] describe a process of translating problems stated in first order logic to the Sicstus constraint solver. This is significantly easier than the translation process we have considered because the source language is less rich than ESSENCE' and

the target language is significantly richer than the input language of MINION. Rafeh *et al.* present the mapping process of Zinc [14], a modelling language, to design models that apply different solving techniques. Though ESSENCE' and Zinc are both on the same level of abstraction, the mapping targets are quite diverse. We would like to incorporate further reformulations into the tailoring process to enhance the final model. We will draw on our work on CGRASS [5] for this, which focused on the reformulation of individual problem instances.

Acknowledgements. Ian Miguel is supported by a UK Royal Academy of Engineering/EPSRC Research Fellowship. Andrea Rendl is supported by a DOC fFORTE scholarship of the Austrian Academy of Sciences and UK EPSRC grant EP/D030145/1. We thank Chris Jefferson for his advice on MINION.

References

1. Charnley, J., Colton, S., Miguel, I.: Automatic generation of implied constraints. In: European Conference on Artificial Intelligence (ECAI), pp. 73–77 (2006)
2. Frisch, A.M., Grum, M., Jefferson, C., Martínez Hernández, B., Miguel, I.: The design of essence: A constraint language for specifying combinatorial problems. In: International Joint Conference on Artificial Intelligence (IJCAI), pp. 80–87 (2007)
3. Frisch, A.M., Hnich, B., Kiziltan, Z., Miguel, I., Walsh, T.: Propagation algorithms for lexicographic ordering constraints. *Artificial Intelligence* 170(10), 803–834 (2006)
4. Frisch, A.M., Jefferson, C., Martínez Hernández, B., Miguel, I.: The rules of constraint modelling. In: International Joint Conference on Artificial Intelligence (IJCAI), pp. 109–116 (2005)
5. Frisch, A.M., Miguel, I., Walsh, T.: CGRASS: A system for transforming constraint satisfaction problems. In: International Workshop on Constraint Solving and Constraint Logic Programming, pp. 15–30 (2002)
6. Gent, I.P., Jefferson, C., Miguel, I.: Minion: A fast scalable constraint solver. In: European Conference on Artificial Intelligence (ECAI), pp. 98–102 (2006)
7. Gent, I.P., Jefferson, C., Miguel, I.: Watched literals for constraint propagation in minion. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 182–197. Springer, Heidelberg (2006)
8. Gent, I.P., Jefferson, C.A., Miguel, I., Petrie, K., Rendl, A.: Minion manual, version 0.4.1., <http://minion.sourceforge.net>
9. Hnich, B., Walsh, T., Smith, B.M.: Dual modelling of permutation and injection problems. *Journal of Artificial Intelligence Research (JAIR)* 21, 357–391 (2004)
10. Prosser, P., Selensky, E.: A study of encodings of constraint satisfaction problems with 0/1 variables. In: International Workshop on Constraint Solving and Constraint Logic Programming, pp. 121–131 (2002)
11. Régim, J.-C.: A filtering algorithm for constraints of difference in cps. In: National Conference on Artificial Intelligence (AAAI), pp. 362–367 (1994)
12. Van Hentenryck, P., Carillon, J.-P.: Generality versus specificity: An experience with AI and OR techniques. In: National Conference on Artificial Intelligence (AAAI), pp. 660–664 (1988)

13. van Hoeve, W.-J., Katriel, I.: Global constraints. In: Handbook of constraint programming, Elsevier (2006)
14. de la Banda, M.G., Marriott, K., Rafah, R., Wallace, M.: From Zinc to Design Model. In: Hanus, M. (ed.) PADL 2007. LNCS, vol. 4354, pp. 215–229. Springer, Heidelberg (2006)
15. Harvey, W., Winterer, T.: Solving the MOLR and Social Golfers Problems. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 286–300. Springer, Heidelberg (2005)

A Meta-CSP Model for Optimal Planning

Peter Gregory, Derek Long, and Maria Fox

University of Strathclyde

Glasgow, UK

{firstname.lastname}@cis.strath.ac.uk

Abstract. One approach to optimal planning is to first start with a sub-optimal solution as a seed plan, and then iteratively search for shorter plans. This approach inevitably leads to an increase in the size of the model to be solved. We introduce a reformulation of the planning problem in which the problem is described as a meta-CSP, which controls the search of an underlying SAT solver. Our results show that this approach solves a greater number of problems than both Maxplan and Blackbox, and our analysis discusses the advantages and disadvantages of searching in the backwards direction.

1 Introduction

Optimal AI planning is a PSPACE-complete problem in general. For many problems studied in the planning literature, the plan optimisation problem has actually been shown to be NP-hard [1,2], whilst the plan existence problem is sometimes only polynomial time. For many years, optimal planning has really referred to Graphplan [3] based planners. Successful optimal planners since Graphplan, have relied on its planning graph structure. Notable examples that have also relied on its search strategy are IPP and STAN [4,5]. SAT based planners, including Blackbox [6], have relied on the planning graph structure, but not the original Graphplan search strategy. These planners convert the planning graph into a SAT model and then allow a SAT solver to search the equivalent SAT instance. One part of the Graphplan search was always kept with these planners, and that was the direction of search. Graphplan constructs the planning graph forwards, until all goals appear non-mutex. It then tests if a solution exists, and if not, then it extends the planning graph by a layer and checks again. This process is repeated until the first satisfiable, and optimal, layer is reached.

Maxplan [7] took a different approach to this idea. It initially finds a suboptimal plan using the planner FF [8], and uses this distance as an initial seed length. It then generates the SAT model for the previous length and tries to find a satisfying assignment. Once a satisfying assignment is found, then the previous layer is searched. This process is repeated until the first non-satisfying layer is found. This model relies on FF finding a solution initially, but since the problem of plan existence is often easier than that of plan optimisation, then it is assumed that if a sub-optimal plan cannot be found, then the chance of finding the optimal one are highly limited. It could be asked as to why we would want to plan in this direction when it inevitably leads to larger models than before. The justification is that it makes better use of the underlying SAT technology.

Namely, learnt clauses can be shared between layers, if they are still in the context of the previous layer when a satisfying assignment is found.

A problem with this approach is that often finding plans at suboptimal lengths means building redundancy into plans. However, discovering these redundant paths in plans is just as hard as finding a path that actually achieves something. If a goal is achieved early in a plan, there is nothing to prevent the SAT solver from reversing its choice and then later re-achieving the same goal. Especially because the SAT solver has no way of distinguishing real actions and *noops* (actions that maintain a fact’s truth between two timesteps). This leads to redundant search, and leads the SAT solver to explore areas of the search space that are not interesting in terms of solving the problem. We introduce a Meta-CSP reformulation of the planning problem in which the variables represent the final achievers of goals. The values in those variables represent the possible final achievers of the goals, as the SAT variables that represent them. We compare the performance of this model with Maxplan and Blackbox. Maxplan provides the best comparison for the meta-CSP model as it uses a traditional SAT encoding with a backwards direction of search, so any performance difference can be directly attributed to the model. The comparison with Blackbox is slightly less straightforward as the direction of search is different. It will provide some comparison between the two directions of search and provide arguments for when each is more effective.

In Section 2 we introduce the planning problem, the planning graph, and the translation of the planning graph into a SAT formulation. The meta-CSP model is described in detail in Section 3, comparing it Maxplan and Blackbox in detail. Section 4 and Section 5 show and analyse extensive empirical results from three different problem domains. Section 6 discusses related work and how performance of the model could be improved in the future.

2 Background

Planning is one of the fundamental problems in Artificial Intelligence. The ability to plan and to reason causally and temporally is one of the key features of intelligent behaviour. The planning problem can be defined in the following way:

Definition 1. A STRIPS planning problem $P = (\mathcal{O}, \mathcal{I}, \mathcal{G})$ has three parts: a set of operators \mathcal{O} , a set of conjoined facts \mathcal{I} that represent the initial state and another set of conjoined facts \mathcal{G} that represent (a partial coverage of) the goal state.

The planning problem is to find a set of actions that transforms the initial state into a state where all of the goal facts are true. An action is a particular instantiation of an operator. An operator itself has three parts: a set of facts *pre* that represent the preconditions that have to be true before an action is executed, a set of facts *add* (commonly known as the add list) which is the set of facts that are added after an action is completed, and a final set of facts *del* which represents the facts that are removed from the state when an action is executed. The facts in the operators contain free variables that have to be instantiated for an action to be performed.

Planning problems in the real world include areas as diverse as robotic control, logistics and airport scheduling. Solutions to planning problems can be of any length.

This is one quality of planning problems that makes them difficult to solve. As the optimal length of a plan isn't known beforehand, a single CSP cannot be constructed that certainly finds a plan. However, a series of CSPs can be solved that eventually find a solution.

A constraint satisfaction problem (CSP) P is defined as a triple, $(X, \mathcal{D}, \mathcal{C})$. X is a finite set of n variables, $X = \{x_1, x_2, \dots, x_n\}$. \mathcal{D} is a finite set of domains, $\mathcal{D} = \{D(x_1), D(x_2), \dots, D(x_n)\}$, such that $D(x_i) = \{v_{i_1}, v_{i_2}, \dots, v_{i_m}\}$ is the finite set of possible values for variable x_i and \mathcal{C} is the set of constraints $\mathcal{C} = \{C_1, C_2, \dots, C_m\}$. A constraint C_i is a relation over a subset of the variables $S_i \subseteq X$ that represents the assignments to the variables in S_i that are legal simultaneously. If $S_i = \{x_{i_1}, \dots, x_{i_l}\}$, then $C_i \subseteq D_{i_1} \times \dots \times D_{i_l}$.

An assignment to a variable is a pair $\langle x_i, v \rangle$ such that $(v \in D(x_i))$, meaning variable x_i is assigned the value v . A solution S to a CSP P is a set of assignments $S = \{\langle x_1, v_1 \rangle, \langle x_2, v_2 \rangle, \dots, \langle x_n, v_n \rangle\}$, such that all constraints in \mathcal{C} are satisfied.

2.1 Graphplan and SAT Planning

The Blackbox planner solves planning problems by translating them into a series of SAT instances. Blackbox uses an intermediary representation in its translation to SAT. This was introduced in 1995 in the Graphplan planner [3], and it is called the planning-graph. It is hard to exaggerate the impact that Graphplan has had on the field of planning. The benefit it gave was that of an exponential space compression of the search-space through the planning-graph structure. The planning-graph construction algorithm is given in Algorithm 1. The planning-graph is a layered graph. It has alternating action layers and fact layers. The first of these is a fact layer, with all of the facts from the initial state. The first fact layer contains all facts that can be achieved by applying actions to the initial state. In addition, at each action layer, there is a special action for each fact in the previous fact layers, called a *noop* (short for no-operation). A noop action maintains the truth of a fact between layers; if no action deletes or adds a fact, then that fact is supported by a noop. The second fact layer contains the union of the first fact layer and the facts achieved by the first action layer. This process continues in the same way from there.

Algorithm 1. The Planning-Graph Construction Algorithm

```

f-layer0 ←  $\mathcal{I}$ 
l ← 0
while Any two goals are mutex or  $f\text{-mutex}(l) \neq f\text{-mutex}(l - 1)$  do
  a-layerl+1 ← actions achievable at f-layerl
  f-layerl+1 ← facts achieved by a-layerl+1
  a-mutex(l + 1) ←  $\{(a_1, a_2) \mid p \in del_{a_1}, (p \in pre_{a_2} \vee p \in add_{a_2}) \cup$ 
     $\{(a_1, a_2) \mid p_1 \in pre_{a_1}, p_2 \in pre_{a_2}, (p_1, p_2) \in f\text{-mutex}(l)\}$ 
  f-mutex(l + 1) ←  $\{(f_1, f_2) \mid \text{all achievers of } f_1 \text{ and } f_2 \text{ are mutex}\}$ 
  l ← l + 1
end while

```

At each layer of the graph, a set of mutual-exclusions (*mutexes* from here) are calculated. These are relationships between two facts or actions, meaning they cannot both be true at the time associated with the level of the planning-graph. For example, the actions *sit down* and *walk away* will always be mutex, as one can't perform both at once. Whether or not other actions such as *walk away* and *swing umbrella* are mutex depends on whether time has passed such that both of their preconditions can be achieved simultaneously. Two facts are mutex if all of their achievers are mutex. Two actions are mutex if either of the following conditions hold:

- One action deletes one of the other's preconditions or effects. This is the case with *sit down* and *walk away* since both have the precondition *standing still*.
- One of the actions has a precondition that is mutex with a precondition of the second action.

When the number of mutexes in two successive layers is equal, it is said that the *fixpoint* is reached. If the graph construction continues until the fixpoint and there are still goals remaining that are mutex, then there can be no solution. Since no more mutexes will be eliminated, then the two goals can never be satisfied at the same time. If at some level l , the goals appear non-mutex with each other, then there may be a solution, and the planning-graph can be searched (we refer to this level as the *first search layer*). Graphplan used a backtracking search, starting with the goals and working backwards through the planning-graph. If no plan is found then the graph is extended to length $l + 1$, and so on. Blackbox provides a translation of the planning-graph into a SAT model. The motivation behind the translation is that the SAT model can then take advantage of any new advances in SAT solving technology, with no extra work required. To convert the planning-graph into CNF, firstly the structure (fact and action layers) has to be represented. And secondly, the constraints (effects and mutexes), also have to be represented. Each fact and action, at every layer, is represented by a SAT variable. If a fact is true at some time point, then the SAT variable representing it is also true. The same is true of actions: if a SAT variable representing an action is true, then that action is part of the plan.

The goals and the initial state can be specified as unit clauses. Each action implies its preconditions: that is, if an action is made true, then its preconditions are forced true. If some action a has $pre = p_1, p_2$ then clauses $(\neg a \vee p_1)$ and $(\neg a \vee p_2)$ are added to the model. Facts also imply that they have an achiever. For all of the achievers of a fact, f including a noop if available ($a_1 \dots a_n$, say), clause $(\neg f \vee a_1 \vee \dots \vee a_n)$ is added. This ensures that each fact is achieved by something at each layer, even if that something is a noop. A mutex between two facts or actions x and y in the planning graph is represented by the clause $(\neg x \vee \neg y)$ in the SAT model.

2.2 Maxplan

Maxplan uses the same encoding as Blackbox, but diverges from the traditional direction of search employed by Graphplan derived planners. Instead of starting search at the first layer that the goals appear non-mutex, Maxplan starts search at a higher layer. It is initialised by a seed plan, generated by the sub-optimal planner FF [8]. Once a plan is found then Maxplan iteratively searches for shorter plans, until the shortest is found.

One advantage to this search strategy over Blackbox is that it is in some sense an “any-time” planning strategy. If it fails at some point due to resource restrictions, then so long as the sub-optimal planner returns a solution, then at least a plan is returned, even if sub-optimal. The planner MaxPlan plans in this way and it retains the same model as BlackBox. It also adds “londex constraints”, these are long-distance mutual exclusion relationships. In the Graphplan model, mutexes only act between facts or actions at the same layer, londex constraints act between different layers, hence long-distance. These constraints are not included in the meta-CSP model.

3 The Meta-CSP Model

The contribution of this work is to introduce a new constraint formulation of the planning problem. This model is a higher level description than the SAT description, it captures a concept that the SAT model does not, that of *final achievement* of a goal. This is important because in a planning problem, the first time a goal is achieved may not be the final time it is achieved. Because of this, once a goal is achieved, the planner has to enforce its noops explicitly until the end of the plan.

Definition 2. *Given a planning problem, $P = (\mathcal{O}, \mathcal{I}, \mathcal{G})$, the meta-CSP encoding is a CSP in which $|X| = |\mathcal{G}|$, where $D_i = \{a \mid g_i \in \text{add}(a)\}$. The only constraints on the model are those implied by the underlying SAT encoding.*

The meta-CSP model has a variable for each goal, $g_i \in \mathcal{G}$, each variable’s domain contains the possible achievers of g_i . These are the actions for which g_i is in the *add list*. An assignment, $\langle x_i, a \rangle$, in the meta-CSP means that a is the final achiever of g_i and no actions that delete g_i may appear later in the plan than a . The meta-CSP solver describes the planning problem on two levels. The first level is the original BlackBox formula, the second is the meta-CSP level, just described, where assignments to variables represent the final achievers of goals.

3.1 Meta-CSP Search

To design a search algorithm, the structure of the model and the analysis that led to its conception have to be considered. The motivation behind this model was an analysis of the backdoors of planning problems at satisfiable levels of the planning-graph. This means that the majority of search should be focused on satisfiable instances. Blackbox searches forwards, exactly like Graphplan, from the fix-point layer forwards to the optimal layer. Planning forwards in this manner seems incompatible with the meta-CSP model since all but the final layer will be unsatisfiable. In rejecting this option, the only alternative is search backwards. From some known satisfiable layer, search for a solution in the previous layer. If no solution exists at some layer then the last plan found must be optimal. When implemented in a modern SAT solver, one of the big advantages to planning in this direction is that learnt clauses stay valid between layers. However, this is only valid if the layers are incrementally decreased. If a binary search were used, for example, then clause learning would have to be disabled. The fact that a planner could search in this backwards direction highlights an important point about planning

research. There is almost without exception an assumption that the problem is satisfiable. If it were not for this assumption, then the idea of planning backwards would be pointless as the level at which to start planning may never be found.

One potential complication is solved by the original Graphplan representation: that of problems not having monotonically satisfiable plan lengths. For example, a problem may have solutions at length five, but not at length six, and then again at length seven. When searching backwards, it may seem intuitive that search would stop when no plan at length six is found (and hence mistakenly inferring that the optimal plan is in fact length seven). However, the noops in the plangraph structure mean that even if there is a plan at an earlier level, but not at the current one, it can be found by enforcing the noops to maintain the state at the end of the plan. The meta-CSP solver searches in the same backwards direction as Maxplan. For this purpose, it is required to know an upper bound for which it is known a plan exists. The chosen method for finding this length is by using the sub-optimal planner FF. FF can find reasonable quality plans very quickly. The length of this plan can be used to initialise backwards search. The seed plan also provides a candidate goal ordering. The order in which the goals in the planning problem are achieved is used as the variable ordering for the meta-CSP.

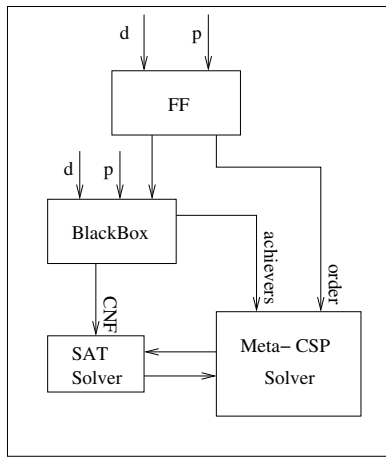


Fig. 1. Architecture of the meta-CSP solver. Input *d* and *p* represent the domain and the problem, respectively.

If the goals contain a necessary goal ordering, the order will be revealed in any valid plan. If the plan is of high quality then it is possible that it reveals important resource allocation and scheduling decisions. This constitutes the seeding process. The next step is the actual search. Although the mechanics of search (propagation, clause learning, etc) still operate in the SAT solver, the key decisions are made at a lifted level in the meta-CSP solver. There is an interleaving between the SAT solver and the meta-CSP solver. The SAT solver will request a decision from the meta-CSP. The meta-CSP solver will then return a decision about what the next choice should be in the SAT solver.

The SAT solver applies this decision and propagate the outcome. There are then two possibilities from here: the first being conflict. In this case, the meta-CSP backtracks, and a new decision is requested. The alternative is that the decision does not lead to conflict, in which case the next decision is requested from the meta-CSP.

During this process, it is possible that the meta-CSP has assigned to all of its variables but the SAT solver still has unassigned variables. In this case, the meta-CSP submits to the default decision-making process of the SAT solver. When the SAT solver backtracks, it triggers a backtrack in the meta-CSP, so long as the decision backtracks as high as the last meta-CSP decision. Clearly, if the meta-CSP returns no solution, then there must be no solution since if no combination of potential goal achievers leads to a successful plan, there must be no plan. The complete architecture of the system is shown in Figure 1. The algorithm relies on three programs: the planner FF, a modified version of Blackbox that outputs both CNF and information about possible goal achievers and a modified version of the zChaff SAT solver, controlled by an added meta-CSP solver. FF provides an upper-bound, l , to plan down from, and a goal ordering. Blackbox then calculates the CNF at length l , and separately outputs the variables that represent:

- The potential final-achievers of each goal and
- the noops that would enforce a particular achieved goal until the end of the plan.

Then the meta-CSP solver orders its variables as they were achieved in the seed plan. It then solves the problem and returns the optimal plan.

4 Results

The following section describes the experimental setup to test the performance of the meta-CSP solver described in Section 3. All of the experiments are performed on a dual-core Intel Pentium D 3.40GHz desktop computer. The meta-CSP solver will be compared against the performance of Maxplan and Blackbox. The performance of each planner will be measured in time taken to solve each problem. The time is taken to be *user-time* + *system-time*, and is limited to a total of 1800 seconds (half of one hour). Although the computer has 2GB of memory, the planners are limited to using 1.5GB. The justification behind this is that if a program approaches 2GB, then CPU usage drops significantly as the planner starts to swap memory. It then becomes very unlikely that either a plan will be returned or that the CPU time will timeout without an unreasonable wait.

Blocksworld, Grid and Driverlog are the three domains that have been selected to test the meta-CSP solver. These will be described in the following sections. Graphs of the results for all of the problem instances are also shown. The graphs are log-scaled on both axes. Since the meta-CSP solver time is the x-axis and the competing planner is the y-axis, any point plotted above the line $y = x$ constitutes a “win” for the meta-CSP solver (as it denotes the competitor took a greater time). To ease the discernment to the eye, the line $y = x$ is also plotted.

4.1 Blocksworld

The Blocksworld problem is a massive part of the planning literature and its history. Although much studied (and occasionally derided), it retains interest because of its interesting structural properties and its simplicity. The form of Blocksworld studied in this work consists of a table on which any number of stacks of blocks can be made. The initial state and goal state are two different configurations of the blocks, the problem is to rearrange the blocks into the goal configuration. The problem is clearly easy to satisfy: one approach could just be to unstack all of the blocks onto the table and then stack them into the goal configuration. However, solving the problem optimally is a difficult task.

Using the problem generator of Thiebaux and Slaney [9], 100 random instances for each size of problem between 5 and 11 blocks were generated, giving 700 instances in total. The results are shown in Figure 2(a) (Maxplan) and in Figure 2(b) (Blackbox). The numbers of unsolved instances are shown in Table 1.

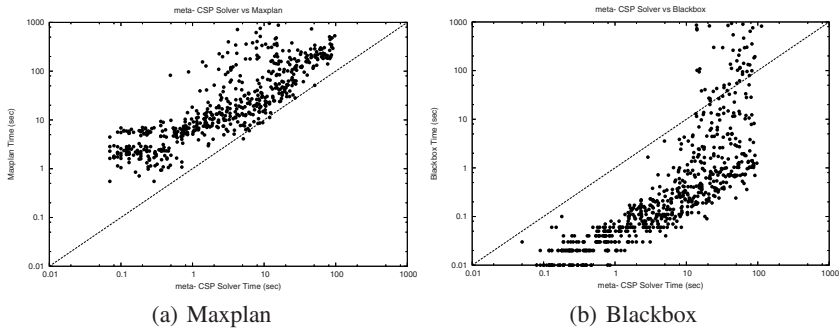


Fig. 2. Combined blocksworld instances

Table 1. Numbers of failed instances in the blocksworld domain. There are 100 problems in each problem set, making 700 in total.

Problem Set	meta-CSP	Blackbox	Maxplan
5 blocks	0	0	4
6 blocks	0	0	0
7 blocks	0	0	0
8 blocks	0	0	0
9 blocks	0	0	0
10 blocks	0	0	54
11 blocks	0	1	52
Total	0	1	110

4.2 Driverlog

Driverlog is a logistics style planning domain, first used in the 2002 International Planning Competition. A Driverlog problem contains four types of object: locations, trucks,

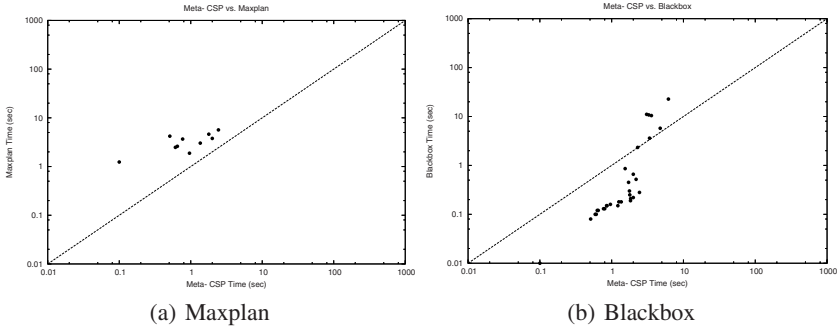


Fig. 3. Combined Driverlog instances

drivers and packages. Locations are connected by roads and paths. Drivers can walk along paths, whilst trucks can drive along roads. However, trucks cannot drive down paths, and drivers cannot walk in the road. Packages have to be transported in trucks, and trucks need to be driven by a driver in order to move anywhere. The typical goal in a Driverlog problem is to move a subset of all drivers, trucks and packages to some locations.

The problem set for the Driverlog domain is generated by a custom generator. This generator differs from the one used in the 2002 IPC in two major respects: it only generates planar graphs for the underlying maps and there is no extra location situated between two locations joined by a path. These changes were made because planar graphs more realistically model a road network and the extra locations vastly increase the number of ground actions when walking actions are not often very important parts of Driverlog plans. The instances generated are all single driver, single truck problems with 15 locations. The parameter to be varied is the number of packages to be delivered. Each package, the driver and the truck each have a goal location, not equal to its start location. The number of packages is varied between 5 and 15, with 10 instances for each size. The results are shown in Figure 3(a) (Maxplan) and in Figure 3(b) (Blackbox). The numbers of unsolved instances are shown in Table 2.

4.3 Grid

The Grid problem has similarities with Driverlog, in that it is a transportation problem. It does differ from it in some crucial ways. The layout of the problem is a grid, with each location connected to adjacent locations up, down, left and right of it in the grid. The cargo in the Grid problem are no longer passive objects whose only role in a plan is to be moved between locations. The cargo in Grid is a number of keys, and to get keys to their desired locations requires different locations to be unlocked. However, the key that unlocks a location may not be the key that needs to be delivered to that same location.

The Grid problem generator is supplied with the FF planning system [8]. The parameters of a Grid problem are x -dimension, y -dimension, number of keys ($k_{\#}$), number of locks ($l_{\#}$) and number of different types of key. Four different size grids will be

Table 2. Numbers of failed instances in the Driverlog domain. There are 10 problems in each problem set, making 150 in total.

Problem Set	meta-CSP	Blackbox	Maxplan
Driverlog 1	0	0	5
Driverlog 2	0	1	7
Driverlog 3	0	2	9
Driverlog 4	0	7	9
Driverlog 5	0	9	10
Driverlog 6	0	10	10
Driverlog 7	0	10	10
Driverlog 8	1	10	10
Driverlog 9	4	10	10
Driverlog 10	2	10	10
Driverlog 11	6	10	10
Driverlog 12	7	10	10
Driverlog 13	6	10	10
Driverlog 14	7	10	10
Driverlog 15	10	10	10
Total	43	119	140

Table 3. Numbers of failed instances in the Grid domain. There are 160 problems in each problem set, making 640 in total.

Problem Set	meta-CSP	Blackbox	Maxplan
grid 3×3	0	8	34
grid 3×4	4	23	51
grid 4×4	4	31	57
grid 4×5	13	40	66
Total	21	102	208

studied, 3×3 , 3×4 , 4×4 , 4×5 . In each of these sizes, there will be ten problems generated for each of the following parameter sets: $\{(k_{\#}, l_{\#}) \mid k_{\#} \in \{1, 2, 3, 4\}, l_{\#} \in \{1, 2, 3, 4\}\}$ giving 640 total instances. The results are shown in Figure 4(a) (Maxplan) and in Figure 4(b) (Blackbox). The numbers of unsolved instances are shown in Table 3.

5 Discussion

Comparing the meta-CSP solver and Maxplan is the best way to evaluate the performance of the meta-CSP model. This is because the major difference between the two planners is which model they use (meta-CSP or purely the SAT encoding). However, for the sake of completeness, it is important to compare the results with another SAT planner. The comparison with Blackbox reveals some curious results, well worthy of discussion. These can provide some explanations of when it is better to use a forwards search direction and when to use a backwards search direction. When viewing the graphs, it

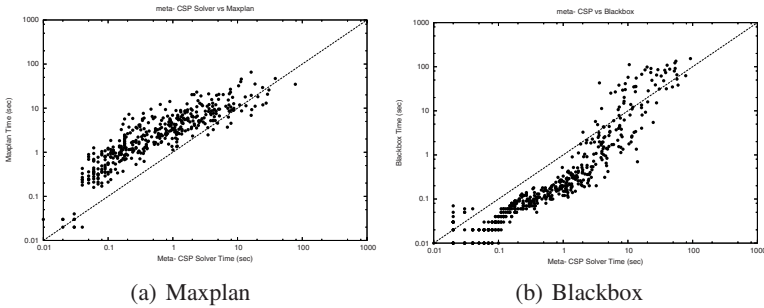


Fig. 4. Combined Grid instances

is important to consider how many problems were unsolved by each planner, as these points cannot be plotted.

5.1 Comparisons with Maxplan

The results were consistently better than those of Maxplan. In all three domains, the meta-CSP solver solves more instances than Maxplan. As shown in Table 1, Table 2 and Table 3, of the 1490 instances, the meta-CSP solver solved 1426 (95.7%) compared to the 458 (30.7%) that Maxplan solved. The graphs clearly show that, except for a few instances, the meta-CSP solver out-performs Maxplan. The graph for the Driverlog instances (Figure 3(a)) is very sparse, and hence not as informative as the other graphs. It does illustrate how few instances Maxplan was able to solve in this domain, however.

In the Driverlog domain, Maxplan failed to solve any instances with 5 or more parcels. For the meta-CSP solver, the first time the solver fails to solve all of the instances was when there were 15 parcels to deliver. To put this in context, problems with 5 parcels are seeded by a plan length of approximately 30 steps, whereas problems with 15 parcels the seed plans are approximately 70 steps. Also in the Blocksworld instances, Maxplan had started to fail 50 percent of the time for number of blocks 10 and 11, where the meta-CSP solver solved every instance. In the Grid problems, both planners suffered difficulties for the more difficult problems, although the meta-CSP solver fared better, solving 619 of the 640 instances, compared to the 432 problems that Maxplan solved. There are several of the harder instances in the Grid domain in which Maxplan performs better than the meta-CSP solver. Any inference that these points indicate that Maxplan is scaling better is incorrect, however, considering the numbers of instances that remain unsolved by Maxplan.

The strongest results were gained in the Driverlog domain. The meta-CSP approach is suited to this domain because firstly, some of the goals are often achievable early in the plan (especially as the numbers of parcels increases). This means that the maintenance of the noops has an important impact on the planning process. Another reason the meta-CSP has an advantage is the fact that the schedule is tight. This means that once a final achiever is decided, propagation is more likely to lead to an outcome than if there was a lot of slack in the problem. Because these problems had only one truck

and one achiever, the decision over resource allocation had already been decided, there was one choice.

As such, the scheduling decisions of when to deliver the packages was the key decision. This is demonstrated in the fact that the meta-CSP solver can solve larger, and more, instances than can Maxplan. In the Grid domain, other decisions that have no relation to the top-level goals are also important and so performance was closer.

5.2 Comparisons with Blackbox

As with Maxplan, Blackbox solved fewer instances than the meta-CSP solver. In total, Blackbox solved 1268 of the 1490 problems. At 85.1%, that is still more than 10% fewer instances solved than the meta-CSP solver. As the graphs illustrate, however, Blackbox usually performs better for the majority of the *solved* instances. There are several reasons that Blackbox performs better. There are overheads that searching in a backwards direction brings with it:

- Before search, the sub-optimal planner that is used to seed the plan has to solve the problem. Although this is typically trivial, it can be an overhead on the total planning time, especially for simple problems, or problems that FF can perform badly on.
- Creating the SAT model is often the dominant part of the meta-CSP search time. Because FF can often overestimate the optimal length greatly, twinned with the fact that in its current implementation, the meta-CSP solver is reliant on Blackbox writing the model to disk (often creating CNF files > 100MB), then just generating the model can take a long time.

It would be disingenuous to leave the discussion there: the difference in performance cannot be entirely down to these overheads. There is a more interesting question of when searching backwards is better than searching forwards. Looking at the individual domains, it can easily be seen that it was in Driverlog that the meta-CSP performs best. Blackbox solved no problems with more than 5 parcels to deliver, the meta-CSP solver even solved some problems with as many as 14 parcels. On the other hand, Blackbox certainly performs better on most Blocksworld instances. The Blocksworld results are split up to show how the planners performed with different numbers of blocks. Notice that although most of the instances that are difficult for Blackbox are of the longest plans, there are some cases where the plans are not that long but still difficult.

The results for 6, 8 and 10 blocks are shown in Figure 5 and they reveal some intriguing behaviour. The variance in the time it takes Blackbox to solve the instances seems to increase with the numbers of blocks, whereas the meta-CSP time seems to remain more stable. With 10 blocks, Blackbox still solves many instances quickly (< 1 second), but others can take several hundred seconds. It seems somewhat intuitive that the meta-CSP solver might solve longer plans faster than Blackbox. This intuition coming from the fact that the meta-CSP solver begins with an overestimate, so a plan close to that estimate is better. The graph of the 11 block instances is plotted again in Figure 6(a), this time annotated with the optimal plan lengths, which range from 5 to 18 steps. To explain this result, it is necessary to also look at a different measure, the distance between the first time the goals appear non-mutex in the planning-graph and the optimal

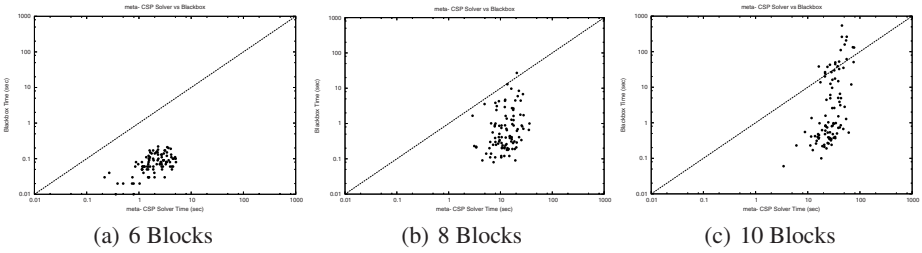


Fig. 5. Blocksworld instances partitioned by size

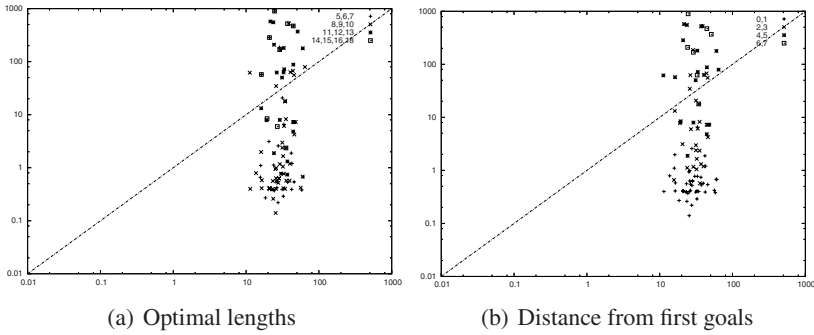


Fig. 6. 11 block annotated Blocksworld instances. The x-axes are the meta-CSP timings, the y-axes represent the Blackbox timings.

plan length. This is shown in Figure 6(b). Notice that it is exactly those instances that are distant from the first search layer, that the meta-CSP solver wins over Blackbox. Only one instance that was in the 6, 7 range was solved faster by Blackbox than by the meta-CSP solver. In the previous range (4, 5), the meta-CSP solver performs better than Blackbox in 14 of 20 instances. There is a large cluster of results that Blackbox solved in the range of 0.1 and 1 seconds that are all solvable either at, or at the next timepoint from, the time the goals first become reachable. So, it seems that problems that are solvable close to the level that the goals first become pairwise non-mutex are best solved by Blackbox, but the more distant problems are best solved by the meta-CSP solver.

One reason this could be is that even when finding sub-optimal plans, the meta-CSP solver is learning information about earlier layers through the learning mechanism of the SAT solver. Blackbox does not remember information between layers, and therefore will often consider the same incorrect choices at each subsequent layer. This result not only provides a good argument for when the meta-CSP solver is better than Blackbox, it suggests an explanation of when planning backwards in the planning-graph is more effective than planning forwards. The best results obtained by the meta-CSP solver were in the Driverlog domain. Driverlog is exactly the type of domain in which all of the goals quickly become pairwise non-mutex, but where the actual optimal level is quite distant from that point.

6 Related Work and Future Work

The idea of solving a problem using a meta-CSP model was used in the BLUEBLOCKER system that optimally solves rectangle packing problems [10]. An interesting feature of this solver is that it models the rectangle packing problem as a disjunctive temporal network, and each assignment in the meta-CSP represents the partial ordering of the rectangles to be packed. Only when a consistent partial ordering has been found, does the system attempt to find a concrete solution, though there may not be a valid solution even then.

Although competitive as a planner, there is clear space for development of the meta-CSP solver. The meta-CSP solver currently has a variable ordering heuristic based on the order of which the goals are achieved in the seed plan. Currently the meta-CSP solver has no value ordering heuristic. Since the variables can have large domains (one goal can have many achievers), an ordering heuristic on the values could provide a performance increase. The meta-CSP model was motivated by the study of backdoors in planning problems. An observation being that the key scheduling decision of when to finally achieve a goal cannot be made explicitly in the SAT solver. The current solver searches across achieving actions of the goals. This means that decisions are also being made about *how* goals are achieved, and not just *when* they are achieved. So, the decisions that the meta-CSP is making are of a very coarse granularity. It may be that the key decisions regard mainly resource allocation, or scheduling, decisions. It has also been noted that some key decisions are allocation decisions that are completely unrelated to the goals (for instance, the keys in the Grid domain). The meta-CSP could be altered such that variables represent this finer-grained structure, simply *when* a goal is achieved, or *which* resources are used in its achievement.

Finally, a closer integration with planning technology could add power to the meta-CSP. One direction the authors would like to pursue is a study into how landmarks analysis [11] could improve search. Landmarks could be added as constraints in the meta-CSP, and because the plan length is bounded, more landmarks could potentially be found.

7 Conclusions

We have introduced a novel reformulation of the planning problem. This is based on a constraint model that searches across the final achievers of the goals of the problem. We have shown this model to be competitive when compared to other state of the art SAT planners. It has been shown to solve more instances than both Maxplan and Blackbox across three domains: Blocksworld, Driverlog and Grid. Time performance is almost always greater than Maxplan on the instances that Maxplan can solve. Often performance is worse than that of Blackbox, but it appears that this has much to do with the distance from the first search layer. Our analysis shows that the performance of Blackbox can degrade when plans are distant from the first time the goals appear non-mutex in the planning-graph. The results also demonstrate the fact that making the key scheduling decision of when to achieve a goal can improve the efficiency of search over a flat SAT model.

References

1. Helmert, M.: Complexity results for standard benchmark domains in planning. *Artificial Intelligence* 143(2), 219–262 (2003)
2. Helmert, M.: New complexity results for classical planning benchmarks. In: *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS 2006)*, pp. 52–61 (2006)
3. Blum, A., Furst, M.: Fast planning through planning graph analysis. In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, pp. 1636–1642 (1995)
4. Nebel, B., Dimopoulos, Y., Koehler, J.: Ignoring Irrelevant Facts and Operators in Plan Generation. In: Steel, S. (ed.) *ECP 1997. LNCS*, vol. 1348, pp. 338–350. Springer, Heidelberg (1997)
5. Long, D., Fox, M.: Efficient implementation of the plan graph in STAN. *Journal of Artificial Intelligence Research* 10, 87–115 (1999)
6. Kautz, H.A., McAllester, D., Selman, B.: Encoding plans in propositional logic. In: *Proceedings of the Fifth International Conference on the Principle of Knowledge Representation and Reasoning (KR'96)*, pp. 374–384 (1996)
7. Xing, Z., Chen, Y., Zhang, W.: Maxplan: Optimal planning by decomposed satisfiability and backward reduction. In: *Proceedings of the Fifth International Planning Competition, International Conference on Automated Planning and Scheduling (ICAPS'06)* (2006)
8. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research (JAIR)* 14, 253–302 (2001)
9. Slaney, J., Thiebaux, S.: Linear time near-optimal planning in the blocks world. In: *AAAI-96. Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, Oregon, USA, pp. 1208–1214. AAAI Press / The MIT Press, Cambridge (1996)
10. Moffitt, M.D., Pollack, M.E.: Optimal Rectangle Packing: A Meta-CSP Approach. In: *Proceedings of the 16th International Conference on Automated Planning and Scheduling*, pp. 93–102 (2006)
11. Hoffmann, J., Porteous, J., Sebastia, L.: Ordered Landmarks in Planning. *Journal of Artificial Intelligence Research (JAIR)* 22, 215–278 (2004)

Reformulation for Extensional Reasoning

Timothy L. Hinrichs and Michael R. Genesereth

Stanford University
{thinrich,genesereth}@cs.stanford.edu

Abstract. Relational databases have had great industrial success in computer science. The power of the paradigm is made clear both by its widespread adoption and by theoretical analysis. Today, automated theorem provers are not able to take advantage of database query engines and therefore do not routinely leverage that source of power. Extensional Reasoning (ER) is an approach to automated theorem proving where the machine automatically translates a logical entailment query into a database, a set of view definitions, and a database query such that the entailment query can be answered by answering the database query. The techniques developed for ER to date are applicable only when the logical theory is axiomatically complete. This paper discusses techniques for reformulating an incomplete theory into a complete theory so that Extensional Reasoning techniques can be applied.

1 Introduction

Relational databases are one of the most successful applications in computer science, yet today's theorem provers fail to capitalize on that success. Extensional Reasoning (ER) [1] is an approach to automated theorem proving where the system automatically transforms a logical entailment query into a database (a set of extensionally defined tables), a set of view definitions (a set of intensionally defined tables), and a database query. Extensional Reasoning has been shown to have powerful properties in the case of axiomatically complete theories, e.g. orders of magnitude performance improvement and more predictable run-time behavior than traditional techniques. This paper examines techniques for Extensional Reasoning that reformulate incomplete theories into complete theories to achieve those same benefits.

Because theory-completion in the context of Extensional Reasoning is performed solely for the sake of efficiency, there are constraints on how the theory can be completed. To preserve soundness and completeness, a theory must be completed so that any entailment query about the original theory can be transformed into an entailment query about the completed theory where the answers to the queries are the same. Handling incomplete theories in Extensional Reasoning therefore requires developing two transformation algorithms: one for the query and one for the theory.

For example, suppose we have the query $\forall x.(p(x) \Rightarrow q(x))$ and the following set of sentences about p and q .

$$\begin{aligned}
&\forall x.(x = a \vee x = b) \\
&a \neq b \\
&\neg p(a) \\
&\neg q(a) \\
&q(b)
\end{aligned}$$

This premise set is incomplete because neither $p(b)$ nor $\neg p(b)$ is entailed; however, the universal query $\forall x.(p(x) \Rightarrow q(x))$ is entailed. For Extensional Reasoning the premise set is converted into a complete theory that consists of two definitions: one for all the possible (consistent) values of p and one for all the possible values of q . The query is then rewritten in terms of those definitions.

$$\forall x.(poss_{p(x)}(x) \Rightarrow poss_{q(x)}(x))$$

$poss_{p(x)}$ is defined so that $poss_{p(x)}(t)$ is true for all those t where $p(t)$ is consistent with the premise set; likewise for $poss_{q(x)}$. Since every ground atom is either consistent with the premises or not, the definitions for $poss_{p(x)}$ and $poss_{q(x)}$ are complete.

$$\begin{aligned}
poss_{p(x)}(x) &\Leftrightarrow (x = b) \\
poss_{q(x)}(x) &\Leftrightarrow (x = b)
\end{aligned}$$

The *poss* version of the query is entailed by the definitions for $poss_{p(x)}$ and $poss_{q(x)}$ if and only if the original entailment query holds. By converting an incomplete theory to a complete theory about *poss* and rewriting the query in terms of *poss*, Extensional Reasoning can be employed to answer entailment queries about incomplete theories. This approach uses what is possible to infer what is necessarily true.

Similar techniques for theory completion (or perhaps more accurately theory minimization) have been studied to great extent in the Nonmonotonic Reasoning literature, e.g. the Closed-World Assumption[2], Predicate Completion[3], Circumscription[4]. Unlike Extensional Reasoning, these settings assume it is acceptable (and in fact desirable) for the reformulation to change the logical consequences of the theory. In contrast, ER completion is consequence-preserving in the sense described above, thus making the Nonmonotonic work insufficient for the purpose of ER.

While we hope Extensional Reasoning will eventually be applied to a wide variety of logics, for the time being we have elected to focus on theories in a decidable logic, placing the issue of efficiency front and center. The particular logic we are studying is a fragment of first-order logic that is a perennial problem in the theorem proving community: it includes the domain closure axiom, which guarantees decidability while allowing arbitrary quantification. This logic, to which the example above belongs, allows us to avoid issues of undecidability at this early stage in the development of Extensional Reasoning, while at the same time giving us the opportunity to make progress on an important class of problems. It should be noted that this logic can be translated into propositional logic, which allows entailment queries to be answered with a SAT solver; however, the naive translation is exponential.

Orthogonal to the choice of logic, Extensional Reasoning can be applied in a variety of settings that differ in the way efficiency is measured. Our work thus far measures efficiency the same way the theorem proving community does. Once the machine has been given a premise set and a query the clock starts; the clock stops once the machine returns an answer. We do not amortize reformulation costs over multiple queries, and we do not assume the premises or query have ever been seen before or will ever be seen again.

This paper presents theory-completion techniques that run in linear or quadratic time and allow Extensional Reasoning to be performed on theories whose incomplete portion is ground. The intent is to use fast reformulation algorithms that put most of the burden for answering the entailment query on industrial-strength database algorithms, thereby leveraging the many man-hours spent developing them. Our technical contributions can be found in the sections on query rewriting (4) and theory completion (5). The remaining sections discuss background (2), examples (3), and future work (6).

2 Background

In our investigations of Extensional Reasoning thus far, the logic we have considered is function-free first-order logic with unique names axioms (UNA) and a domain closure axiom (DCA). The UNA state that every pair of distinct object constants in the vocabulary is unequal. The DCA states that every object in the universe must be one of the object constants in the vocabulary. Together, the UNA and DCA ensure that the only models that satisfy a given set of sentences are the Herbrand models of those sentences, and because the language is function-free, every such model has a finite universe. We call this logic Finite Herbrand Logic (FHL). It is noteworthy that entailment in FHL is decidable.

Besides the existence of UNA and a DCA, the definitions for FHL are the same as those for function-free first-order logic. Terms and sentences are defined as usual. We say a sentence is *closed* whenever it has no free variables, and an *open* sentence has at least one free variable. The definition for a model is the usual one, but because all the models of FHL are finite Herbrand models, it is often convenient to treat a model as a set of ground atoms. When we do that, satisfaction is defined as follows.

Definition 1 (FHL Satisfaction). *The definition for the satisfaction of closed sentences, where the model M is represented as a set of ground atoms, is as follows.*

- $\models_M s = t$ if and only if s and t are syntactically identical.
- $\models_M p(t_1, \dots, t_n)$ if and only if $p(t_1, \dots, t_n) \in M$
- $\models_M \neg\psi$ if and only if $\not\models_M \psi$
- $\models_M \psi_1 \wedge \psi_2$ if and only if $\models_M \psi_1$ and $\models_M \psi_2$
- $\models_M \forall x.\psi(x)$ if and only if $\models_M \psi(a)$ for every object constant a .

An open sentence $\phi(x_1, \dots, x_n)$ with free variables x_1, \dots, x_n is satisfied by M if and only if $\forall x_1 \dots x_n. \phi(x_1, \dots, x_n)$ is satisfied by M according to the above definition.

A set of sentences in FHL constitutes an incomplete theory whenever there is more than one Herbrand model that satisfies it. A set of sentences in FHL constitutes a complete theory whenever there is at most one Herbrand model that satisfies it. Logical entailment is defined as usual: $\Delta \models \phi$ in FHL if and only if every model that satisfies Δ also satisfies ϕ .

In this paper, we demonstrate how to transform an entailment query about an incomplete theory into an entailment query about a complete theory, with the eventual goal of transforming the complete theory into a database system. Consequently, the complete theories we will be targeting have a specific form that makes conversion to a database system straightforward. Every set of sentences constituting a complete theory in this paper will be a set of biconditional definitions such that no predicate is ever (transitively) defined in terms of itself, i.e. a set of nonrecursive biconditional definitions.

Definition 2 (Biconditional Definition). *A biconditional definition is a sentence of the form $r(\bar{x}) \Leftrightarrow \phi(\bar{x})$, where r is a predicate, \bar{x} is a nonrepeating sequence of variables, and $\phi(\bar{x})$ is a sentence with free variables \bar{x} . We refer to $r(\bar{x})$ as the head and $\phi(\bar{x})$ as the body.*

Definition 3 (Nonrecursive biconditional definitions). *A set of biconditional definitions Δ is nonrecursive if and only if the graph $\langle V, E \rangle$ is acyclic.*

V : the set of predicates in Δ

E : $\langle p, q \rangle$ is a directed edge if and only if there is a biconditional in Δ with p in the head and q in the body.

Theorem 1 (Biconditional Completeness [1]). *Suppose Δ is a finite set of nonrecursive, biconditional definitions in FHL, where there is exactly one definition for each predicate in Δ and no definition for $=$. Δ is complete.*

3 Example

In the case of complete theories, Extensional Reasoning can produce substantial speedups when compared to traditional techniques. Consider the following set of nonrecursive biconditional definitions that state which of the squares in Fig. 1 are located to the west of which other squares.

$$\text{westof}(x, y) \Leftrightarrow (\text{duewest}(x, y) \vee \exists z. (\text{samecolumn}(x, z) \wedge \text{duewest}(z, y)))$$

$$\text{samecolumn}(x, y) \Leftrightarrow (\text{duenorth}(x, y) \vee \text{duenorth}(y, x))$$

$$\text{duewest}(x, y) \Leftrightarrow \left(\begin{array}{l} \text{onewest}(x, y) \vee \\ \exists z. (\text{onewest}(x, z) \wedge \text{onewest}(z, y)) \vee \\ \exists zw. (\text{onewest}(x, z) \wedge \text{onewest}(z, w) \wedge \text{onewest}(w, y)) \end{array} \right)$$

a	b	c	d
*	f	g	h
i	j	k	≈
m	n	o	p

Fig. 1. A Wumpus World snapshot with a shine to the west and a stench to the east

$$\begin{aligned}
 \text{duenorth}(x, y) &\Leftrightarrow \left(\begin{array}{l} \text{onenorth}(x, y) \vee \\ \exists z.(\text{onenorth}(x, z) \wedge \text{onenorth}(z, y)) \vee \\ \exists zw.(\text{onenorth}(x, z) \wedge \text{onenorth}(z, w) \wedge \text{onenorth}(w, y)) \end{array} \right) \\
 \text{onewest}(x, y) &\Leftrightarrow \left(\begin{array}{l} (x = a \wedge y = b) \vee (x = b \wedge y = c) \vee (x = c \wedge y = d) \vee \\ (x = e \wedge y = f) \vee (x = f \wedge y = g) \vee \dots \end{array} \right) \\
 \text{onenorth}(x, y) &\Leftrightarrow \left(\begin{array}{l} (x = a \wedge y = e) \vee (x = e \wedge y = i) \vee (x = i \wedge y = m) \vee \\ (x = b \wedge y = f) \vee (x = f \wedge y = j) \vee \dots \end{array} \right)
 \end{aligned}$$

Translating this set of biconditionals into a database system requires linear time. We compared the time required to answer the query $\text{westof}(a, p)$ using the axioms as written above and Prover9¹, Otter’s successor, with the database version and a homegrown implementation of top-down datalog^- [5]. Prover9 found a proof in about 300 seconds, and the datalog^- implementation found an answer in 0.017 seconds (including translation), a speedup factor of 10^4 . We also compared times for the query $\text{westof}(a, m)$, which is not entailed. Prover9 ran for over 30 minutes without finishing, and the datalog^- implementation finished in 0.022 seconds, a speedup factor of at least 10^5 .

Answering the database version of the query can sometimes be much faster because the knowledge captured in the classical logic sentences has been organized into a special form that is amenable to processing. In a database system, every predicate corresponds to exactly one table, and the system differentiates between the explicit (extensional) predicates and the implicit (intensional) predicates. Extensionally defined tables can be reasoned about very efficiently because of smart indexing; intensionally defined tables can be reasoned about efficiently because negation as failure is used.

Thus, using the techniques introduced in [1], complete theories can be reasoned about very efficiently using database systems. However, if one were to add even just a small amount of incompleteness into a complete theory by extending the theory with new predicates, those techniques could no longer be applied. This is unfortunate since the speedups in the complete case seem to be large enough to absorb some extra overhead for dealing with incomplete theories.

¹ <http://www.cs.unm.edu/~mccune/prover9/>

For example, suppose we add to the theory above the following set of sentences.

$$\begin{aligned} &gold(x) \wedge gold(y) \Rightarrow x = y \\ &gold(a) \vee gold(f) \vee gold(i) \\ &wumpus(x) \wedge wumpus(y) \Rightarrow x = y \\ &wumpus(h) \vee wumpus(k) \vee wumpus(p) \end{aligned}$$

Together these axioms describe the snapshot of Wumpus World shown in Fig. 1, where an agent has visited two locations. Recall that the Wumpus World contains a hidden pile of gold, the object that is sought, and a hidden wumpus, the beast to be avoided. The percepts sensed in the western cell indicate the gold is in cell a , f , or i , and the percepts in the eastern cell indicate the wumpus is in cell h , k , or p . Consider then the entailment query, “Is the gold situated to the west of the wumpus?”

$$\forall xy.(gold(x) \wedge wumpus(y) \Rightarrow westof(x, y))$$

Clearly the answer is yes since each of the possible gold locations (a , f , and i) is situated to the west of all the possible wumpus locations (h , k , and p).

To answer this query using Extensional Reasoning techniques, we must construct a complete theory that faithfully represents the original. Our approach is to construct complete definitions for new predicates that represent all the possible values for the incomplete predicates. The entailment query is rewritten so that it is expressed in terms of these new predicates.

Consider just the sentences involving *gold*.

$$\begin{aligned} &gold(x) \wedge gold(y) \Rightarrow x = y \\ &gold(a) \vee gold(f) \vee gold(i) \end{aligned}$$

We construct a predicate, $poss_{gold(x)}(x)$ that represents all the gold tuples that are consistent with the *gold* sentences above. In this case, $gold(a)$, $gold(f)$, and $gold(i)$ are the only consistent *gold* atoms.

$$poss_{gold(x)}(x) \Leftrightarrow (x = a \vee x = f \vee x = i)$$

Similar reasoning concludes that $poss_{wumpus(x)}$ is true of h , k , and p .

$$poss_{wumpus(y)}(y) \Leftrightarrow (y = h \vee y = k \vee y = p)$$

The original entailment query, “Is the gold situated to the west of the wumpus?” is written in terms of $poss_{gold(x)}$ and $poss_{wumpus(y)}$.

$$\forall xy.(poss_{gold(x)}(x) \wedge poss_{wumpus(y)}(y) \Rightarrow westof(x, y))$$

Because this query is about a complete theory (the biconditional definitions for *westof*, and the two biconditionals for $poss_{gold(x)}$ and $poss_{wumpus(x)}$), Extensional Reasoning techniques can be applied to answer the query using a database system. We compared the time required to answer the query using the

westof, *gold*, and *wumpus* axioms and Prover9 with the *westof*, $poss_{gold(x)}$, and $poss_{wumpus(y)}$ axioms and our *datalog*[⊥] implementation. Prover9 found a proof in about 350 seconds, and the *datalog*[⊥] implementation found an answer in 0.050 seconds, a speedup factor of 10^3 . For the query, “Is the wumpus situated to the west of the gold?”, which is not entailed, Prover9 ran for over 35 minutes without finishing, and the *datalog*[⊥] implementation finished in 0.020 seconds, a speedup factor of at least 10^5 .

This transformation of an incomplete theory to a complete *poss* theory can be applied in general settings having nothing to do with the Wumpus World. This paper introduces techniques for performing that transformation, using the Wumpus World to illustrate throughout.

4 Reformulating the Query

Satisfiability has long been used in refutational theorem proving to answer logical entailment queries: $\Delta \models \phi$ if and only if $\Delta \cup \{\neg\phi\}$ is unsatisfiable. Satisfiability is usually leveraged solely at the meta-level by people who are trying to justify the soundness and completeness of proof systems. In contrast, the techniques presented here use satisfiability within the logical theory, resulting in proof systems reasoning about satisfiability directly. The central activities required for this approach are the construction and manipulation of definitions for new predicates, each of which represents the satisfiability of a particular sentence in combination with a background theory. $poss_\phi$, a predicate representing the satisfiability of sentence ϕ , is true if and only if ϕ is consistent with the theory Δ . When $\phi(\bar{x})$ has n free variables \bar{x} , its *poss* predicate takes n arguments and is true of \bar{t} exactly when $\phi(\bar{t})$ is consistent with Δ .²

For example, suppose we wanted to know whether the following sentence were consistent with the Wumpus World theory illustrated in Sect. 3.

$$\exists xy.(gold(x) \wedge wumpus(y) \wedge \neg westof(x, y))$$

In the Extensional Reasoning approach, we would first construct a definition for the 0-ary predicate

$$poss_{\exists xy.(gold(x) \wedge wumpus(y) \wedge \neg westof(x, y))}$$

then convert that definition into a database system, and finally determine satisfiability by posing the database query $poss_{\exists xy.(gold(x) \wedge wumpus(y) \wedge \neg westof(x, y))}$.

This particular satisfiability query is relevant to the Wumpus World example from the last section because its negation is the entailment query posed there, “Is the gold situated to the west of the wumpus?” Thus, if the above query is inconsistent with the premises then the entailment query must be true. Because

² In this paper, instead of writing out n variables as x_1, \dots, x_n , we will write \bar{x} ; likewise terms t_1, \dots, t_n will be written \bar{t} . Sometimes a predicate will take some number of variables that we will want to group together; we will use similar shorthand. For example, $p(x_1, \dots, x_n, y_1, \dots, y_m)$ will be written $p(\bar{x}, \bar{y})$.

the definition for $poss_\phi$ is always complete, the above query is inconsistent if and only if the following is true.

$$\neg POSS \exists xy.(gold(x) \wedge wumpus(y) \wedge \neg westof(x,y))$$

More generally, $\Delta \models \phi$ if and only if $\neg poss_{\neg\phi}$ is true. This conversion of an entailment query into a $poss$ query is thus straightforward and can be applied to every query, but the construction of $poss$ definitions is more complicated. The remainder of this section demonstrates how to simplify a $poss$ query to make constructing its definition easier.

Formally, the algorithm that constructs $poss$ definitions, which we call $poss$ -CONSTRUCTION, must abide by the following constraints.

Definition 4 (poss-construction). $poss$ -CONSTRUCTION $[\phi(\bar{x}), \Delta]$ is an operation on a sentence $\phi(\bar{x})$ with free variables \bar{x} and a set of sentences Δ that produces a typed, complete definition for $poss_{\phi(\bar{x})}(\bar{x})$ such that $poss_{\phi(\bar{x})}(\bar{t})$ is true if and only if $\phi(\bar{t})$ is consistent with Δ . $poss_{\phi(\bar{x})}(\bar{x})$ is typed so that it only applies to object constants in Δ .

Note that $poss_\phi$ and $poss_\psi$ are logically equivalent if ϕ and ψ are logically equivalent. This is the property of $poss$ discussed earlier that allows us to equate the two predicates shown below.

$$\begin{aligned} &\neg POSS \neg \forall xy.(gold(x) \wedge wumpus(y) \Rightarrow westof(x,y)) \\ \Leftrightarrow &\neg POSS \exists xy.(gold(x) \wedge wumpus(y) \wedge \neg westof(x,y)) \end{aligned}$$

As a rule of thumb, the definition for $poss_\phi$ is easier to construct the shorter the sentence ϕ . Fortunately, $poss_\phi$ can sometimes be broken up into predicates $poss_{\phi_1}, \dots, poss_{\phi_n}$ so that ϕ_i is shorter than ϕ for every i . For example, it turns out that $poss$ can be distributed across existential quantifiers: $poss_{\exists x.\phi(x)}$ is equivalent to $\exists x. poss_{\phi(x)}(x)$.

Theorem 2 (poss distributes over \exists). In FHL, $poss_{\exists x.\phi(x,\bar{y})}(\bar{y})$ is logically equivalent to $\exists x. poss_{\phi(x,\bar{y})}(x, \bar{y})$.

Proof. (\Rightarrow) Suppose $poss_{\exists x.\phi(x,\bar{y})}(\bar{t})$ is true. Then there is some model M that satisfies $\exists x.\phi(x,\bar{t})$, which ensures there is some object constant u such that $\models_M \phi(u,\bar{t})$ since Δ is in FHL. By the definition of $poss$, $poss_{\phi(x,\bar{y})}(u,\bar{t})$ must therefore be true, which implies $\exists x. poss_{\phi(x,\bar{y})}(x,\bar{t})$ must be true.

(\Leftarrow) Suppose $\exists x. poss_{\phi(x,\bar{y})}(x,\bar{t})$ is true. Then because of FHL semantics and the completeness of $poss$ definitions, there is some u such that $poss_{\phi(x,\bar{y})}(u,\bar{t})$ is true. By the definition of $poss$, there must be some model that satisfies $\phi(u,\bar{t})$. That same model certainly satisfies $\exists x.\phi(x,\bar{t})$, and serves as a witness for the fact that $poss_{\exists x.\phi(x,\bar{y})}(\bar{t})$ is true.

Because we have shown equivalence for all instances of the theorem, by Herbrand semantics, we have proven the theorem. \square

Continuing the Wumpus World example, we can distribute *poss* across the existential quantifier.

$$\begin{aligned} & \neg \text{poss} \exists xy. (\text{gold}(x) \wedge \text{wumpus}(y) \wedge \neg \text{westof}(x, y)) \\ \Leftrightarrow & \neg \exists xy. \text{poss} \text{gold}(x) \wedge \text{wumpus}(y) \wedge \neg \text{westof}(x, y)(x, y) \end{aligned}$$

Because *poss* distributes across existential quantifiers, it is not surprising that it distributes over disjunction because in FHL the two are interconvertible.

Theorem 3 (*poss* distributes over \vee). *In FHL, $\text{poss}_{\phi(\bar{x}, \bar{y}) \vee \psi(\bar{x}, \bar{z})}(\bar{x}, \bar{y}, \bar{z})$ is logically equivalent to $\text{poss}_{\phi(\bar{x}, \bar{y})}(\bar{x}, \bar{y}) \vee \text{poss}_{\psi(\bar{x}, \bar{z})}(\bar{x}, \bar{z})$.*

Proof. (\Rightarrow) Suppose $\text{poss}_{\phi(\bar{x}, \bar{y}) \vee \psi(\bar{x}, \bar{z})}(\bar{t}_1, \bar{t}_2, \bar{t}_3)$ is true. Then there is some model M that satisfies $\phi(\bar{t}_1, \bar{t}_2) \vee \psi(\bar{t}_1, \bar{t}_3)$. By the definition of disjunction, M must therefore satisfy either disjunct or both disjuncts. If the model satisfies only one of the disjuncts, then certainly *poss* for that disjunct is true, ensuring the disjunction of the two *poss* statements is true. If the model satisfies both, then the conjunction of the two *poss* statements is true, which ensures their disjunction is also true. In either case, $\text{poss}_{\phi(\bar{x}, \bar{y})}(\bar{t}_1, \bar{t}_2) \vee \text{poss}_{\psi(\bar{x}, \bar{z})}(\bar{t}_1, \bar{t}_3)$ is true.

(\Leftarrow) Suppose $\text{poss}_{\phi(\bar{x}, \bar{y})}(\bar{t}_1, \bar{t}_2) \vee \text{poss}_{\psi(\bar{x}, \bar{z})}(\bar{t}_1, \bar{t}_3)$ is true. Then either there is a model that satisfies $\phi(\bar{t}_1, \bar{t}_2)$ or there is a model that satisfies $\psi(\bar{t}_1, \bar{t}_3)$ or there is a model that satisfies both. In the first case, since the model satisfies $\phi(\bar{t}_1, \bar{t}_2)$, it must satisfy every disjunction including it as a disjunct. That model must therefore satisfy $\phi(\bar{t}_1, \bar{t}_2) \vee \psi(\bar{t}_1, \bar{t}_3)$. Likewise for the second case. The third case guarantees the existence of a model that satisfies the conjunction and therefore the disjunction. Therefore, in all three cases, we know that there is a model that satisfies $\phi(\bar{t}_1, \bar{t}_2) \vee \psi(\bar{t}_1, \bar{t}_3)$, which means $\text{poss}_{\phi(\bar{x}, \bar{y}) \vee \psi(\bar{x}, \bar{z})}(\bar{t}_1, \bar{t}_2, \bar{t}_3)$ is true.

Because we have shown equivalence for all instances of the theorem, by Herbrand semantics, we have proven the theorem. \square

It turns out that *poss* does not always distribute over universal quantifiers, conjunction, or negation. If *poss* did distribute over negation, it would also distribute over universal quantifiers and conjunction; hence, the counterexample of negation is the most important. Consider the propositional sentence p in a theory where both p and $\neg p$ are consistent. In this theory both $\text{poss}_{\neg p}$ and poss_p are true. If *poss* could be distributed over \neg , then $\text{poss}_{\neg p}$ implies $\neg \text{poss}_p$, which contradicts the fact that poss_p is true.

Nevertheless, when comparing the Wumpus World *poss* query above with a logically equivalent rewriting of the query from Sect. 3, shown below, we see that *poss* can be distributed over conjunction in certain circumstances.

$$\neg \exists xy. (\text{poss}_{\text{gold}(x)}(x) \wedge \text{poss}_{\text{wumpus}(y)}(y) \wedge \neg \text{westof}(x, y))$$

Intuitively, the possible locations for the gold are entirely independent of the possible locations for the wumpus. This intuition is materialized syntactically by the fact that the sentences constraining *gold* never mention *wumpus*, and the sentences constraining *wumpus* never mention *gold*. Thus if $\text{gold}(t)$ were satisfied by some model and $\text{wumpus}(u)$ were satisfied by some model, there

must be some model that satisfies $gold(t) \wedge wumpus(u)$. We say that two such sentences are *independent*.

Definition 5 (Sentence Independence). *Let Δ be a sentence set in FHL and $\phi(\bar{x}, \bar{z})$ and $\psi(\bar{y}, \bar{z})$ be sentences whose vocabularies are subsets of the vocabulary of Δ that share the variables \bar{z} . $\phi(\bar{x}, \bar{z})$ and $\psi(\bar{y}, \bar{z})$ are independent with respect to Δ if and only if for every tuple of object constants \bar{t} , \bar{u} , and \bar{v} , if $\phi(\bar{t}, \bar{v})$ is consistent with Δ and $\psi(\bar{u}, \bar{v})$ is consistent with Δ then $\phi(\bar{t}, \bar{v}) \wedge \psi(\bar{u}, \bar{v})$ is consistent with Δ .*

Whenever two sentences are independent, *poss* distributes over their conjunction.

Theorem 4 (*poss* Distributes over Conjunctions of Independent Sentences). *In FHL, suppose $\phi(\bar{x}, \bar{z})$ and $\psi(\bar{y}, \bar{z})$ are independent with respect to Δ . Then $poss_{\phi(\bar{x}, \bar{z}) \wedge \psi(\bar{y}, \bar{z})}(\bar{x}, \bar{y}, \bar{z})$ is equivalent to $poss_{\phi(\bar{x}, \bar{z})}(\bar{x}, \bar{z}) \wedge poss_{\psi(\bar{y}, \bar{z})}(\bar{y}, \bar{z})$.*

Proof. (\Rightarrow) Suppose $poss_{\phi(\bar{x}, \bar{z}) \wedge \psi(\bar{y}, \bar{z})}(\bar{t}, \bar{u}, \bar{v})$ is true. Then there is some model that satisfies $\phi(\bar{t}, \bar{v}) \wedge \psi(\bar{u}, \bar{v})$. That same model must then also satisfy $\phi(\bar{t}, \bar{v})$ and $\psi(\bar{u}, \bar{v})$ individually, ensuring $poss_{\phi(\bar{x}, \bar{z})}(\bar{t}, \bar{v})$ and $poss_{\psi(\bar{y}, \bar{z})}(\bar{u}, \bar{v})$ are both true. Thus, their conjunction is true.

(\Leftarrow) Suppose $poss_{\phi(\bar{x}, \bar{z})}(\bar{t}, \bar{v}) \wedge poss_{\psi(\bar{y}, \bar{z})}(\bar{u}, \bar{v})$ is true. Then there must be some model that satisfies $\phi(\bar{t}, \bar{v})$ and some model that satisfies $\psi(\bar{u}, \bar{v})$. By the independence of $\phi(\bar{x}, \bar{z})$ and $\psi(\bar{y}, \bar{z})$, since $\phi(\bar{t}, \bar{v})$ is consistent with Δ and $\psi(\bar{u}, \bar{v})$ is consistent with Δ , $\phi(\bar{t}, \bar{v}) \wedge \psi(\bar{u}, \bar{v})$ is consistent with Δ . Hence, $poss_{\phi(\bar{x}, \bar{z}) \wedge \psi(\bar{y}, \bar{z})}(\bar{t}, \bar{u}, \bar{v})$ is true.

Because we have shown equivalence for all instances of the theorem, by Herbrand semantics, we have proven the theorem. □

In light of the last theorem, independence is an important property. One sufficient condition for determining whether $p(\bar{x}, \bar{z})$ and $q(\bar{y}, \bar{z})$ are independent is syntactic: if the constraints on two predicates never (transitively) mention the other predicate (where predicates with complete definitions can be ignored) then the two predicates are independent. Formally, this condition can be defined by examining the dependency graph of a sentence set.

Definition 6 (Undirected Dependency Graph). *An Undirected Dependency Graph for a set of sentences Δ is a graph $\langle V, E \rangle$.*

- $u \in V$ if and only if u is a predicate in Δ without a complete definition
- The undirected edge (u, v) is in E if and only if there is some sentence in Δ that uses both the predicates u and v .

Theorem 5 (Syntactic Independence). *Let Δ be a satisfiable set of FHL sentences, and let G be its Undirected Dependency Graph. Suppose that in G each of the predicates used in $\phi(\bar{x}, \bar{z})$ is in a different connected component than every predicate in $\psi(\bar{y}, \bar{z})$. Then $\phi(\bar{x}, \bar{z})$ and $\psi(\bar{y}, \bar{z})$ are independent with respect to Δ .*

Connected components can be computed in time linear in the size of the Undirected Dependency Graph, and this graph is linear in the size of Δ ³. Given the components, checking syntactic independence of two sentences is linear in their size.

Going back to the Wumpus World example, syntactic independence guarantees that *gold*, *wumpus*, and *westof* are all independent. This pushes the simplification one step further than before.

$$\begin{aligned} & \neg \exists xy. \text{poss}_{\text{gold}(x) \wedge \text{wumpus}(y) \wedge \neg \text{westof}(x,y)}(x, y) \\ \Leftrightarrow & \neg \exists xy. (\text{poss}_{\text{gold}(x)}(x) \wedge \text{poss}_{\text{wumpus}(y)}(y) \wedge \text{poss}_{\neg \text{westof}(x,y)}(x, y)) \end{aligned}$$

The final step requires one more theorem. *westof* is a predicate with a complete definition. Just as independence affects the properties of *poss*, completeness affects those properties as well, and to an even greater extent. (1) A sentence that is complete is independent of every other sentence. (2) poss_ϕ is logically equivalent to ϕ when ϕ is complete.

Theorem 6 (Completeness Guarantees Independence). *Let Δ be a satisfiable set of FHL sentences complete with respect to $\phi(\bar{x})$, i.e. for every tuple of object constants \bar{t} , $\Delta \models \phi(\bar{t})$ or $\Delta \models \neg\phi(\bar{t})$. Then $\phi(\bar{x})$ is independent of every other sentence ψ with respect to Δ .*

Proof. Suppose that both $\phi(\bar{t})$ and ψ are individually consistent with Δ . Then there must be a model M that satisfies $\phi(\bar{t})$ and a model N that satisfies ψ . Because Δ entails either $\phi(\bar{t})$ or $\neg\phi(\bar{t})$ and M satisfies $\phi(\bar{t})$, Δ must entail $\phi(\bar{t})$ (for M is a countermodel to $\Delta \models \neg\phi(\bar{t})$). Therefore, every model that satisfies Δ must satisfy $\phi(\bar{t})$, including N . Hence, N must satisfy both $\phi(\bar{t})$ and ψ . Since the argument was made for an arbitrary \bar{t} , it holds for all \bar{t} , and by Herbrand semantics, $\phi(\bar{x})$ and ψ must therefore be independent wrt Δ . \square

Theorem 7 (Completeness and *poss* Redundancy). *Let Δ be a satisfiable set of FHL sentences complete with respect to $\phi(\bar{x})$, i.e. for all tuples \bar{t} of object constants in Δ , Δ entails $\phi(\bar{t})$ or Δ entails $\neg\phi(\bar{t})$. Then*

$$\Delta \cup \{\text{poss-CONSTRUCTION}[\phi(\bar{x}), \Delta]\} \models \text{poss}_{\phi(\bar{x})}(\bar{x}) \Leftrightarrow \phi(\bar{x})$$

Proof. (\Rightarrow) Suppose $\text{poss}_{\phi(\bar{x})}(\bar{t})$ is true. Then there is some model M that satisfies $\phi(\bar{t})$. Because Δ entails either $\phi(\bar{t})$ or its negation, Δ must entail $\phi(\bar{t})$ since in the other case M is a countermodel.

(\Leftarrow) Suppose $\Delta \models \phi(\bar{t})$. Then because Δ is satisfiable, there is some model that satisfies Δ and, because entailment holds, that model satisfies $\phi(\bar{t})$. Thus, $\text{poss}_{\phi(\bar{x})}(\bar{t})$ is true.

Because we have shown equivalence for all instances of the theorem, by Herbrand semantics, we have proven the theorem. \square

³ Technically, an Undirected Dependency Graph might require a quadratic number of edges but there is a linear graph sufficient for the purpose of computing connected components.

Applying these two theorems to simplify a *poss* query requires knowing which predicates have complete definitions in the theory. The algorithm investigated in [1] detects whether an entire theory is complete using a variation of the well-known CFG algorithm for checking emptiness. A simple variant of that procedure is an anytime algorithm that attempts to find the predicates in a theory that have complete definitions and when run to completion costs low-order polynomial time in the size of the theory.

In the context of Wumpus World, the results above guarantee that (1) *westof* is independent of every other sentence and (2) $\text{poss}_{\neg\text{westof}(x,y)}(x,y)$ is logically equivalent to $\neg\text{westof}(x,y)$. Using all the theorems in this section, we see the following sequence of *poss* simplifications of the Wumpus World query.

$$\begin{aligned}
 \Delta &\models \forall xy.(gold(x) \wedge wumpus(y) \Rightarrow westof(x,y)) \\
 &\Leftrightarrow \neg\text{poss}_{\neg\forall xy.(gold(x)\wedge wumpus(y)\Rightarrow westof(x,y))}(\text{poss Semantics}) \\
 &\Leftrightarrow \neg\text{poss}_{\exists xy.(gold(x)\wedge wumpus(y)\wedge\neg westof(x,y))}(\text{Log. Equiv.}) \\
 &\Leftrightarrow \neg\exists xy.\text{poss}_{gold(x)\wedge wumpus(y)\wedge\neg westof(x,y)}(x,y) \text{ (Distr. over } \exists) \\
 &\Leftrightarrow \neg\exists xy.(\text{poss}_{gold(x)}(x) \wedge \text{poss}_{wumpus(y)}(y) \wedge \text{poss}_{\neg westof(x,y)}(x,y)) \text{ (Distr. over } \wedge) \\
 &\Leftrightarrow \neg\exists xy.(\text{poss}_{gold(x)}(x) \wedge \text{poss}_{wumpus(y)}(y) \wedge \neg\text{westof}(x,y)) \text{ (poss Redundancy)} \\
 &\Leftrightarrow \forall xy.(\text{poss}_{gold(x)}(x) \wedge \text{poss}_{wumpus(y)}(y) \Rightarrow \text{westof}(x,y)) \text{ (Log. Equiv.)}
 \end{aligned}$$

As demonstrated in this section, any logical entailment query can be written as a *poss* query, and a *poss* query can sometimes be simplified by distributing *poss* over logical connectives, thereby reducing the complexity of constructing *poss* definitions. With the exception of the completeness computation, the simplifications take time linear in the size of the *poss* sentence after spending time linear in the size of the theory to compute which predicates are independent.

5 Completing the Theory

The previous section demonstrated how to transform an entailment query into a *poss* query and enumerated some of the properties of *poss* that allow such a query to be simplified. That simplification is important because it makes constructing definitions for *poss* more efficient. Constructing *poss* definitions is a form of theory-completion because every proper *poss* definition is complete: every sentence is either consistent with a background theory or it is inconsistent with that theory. This section demonstrates how to construct *poss* definitions for ground clauses, where the cost of reformulation is quadratic.

Consider a set of ground clauses that constrain a single predicate *p*, where we use $[\neg]p(\dots)$ to indicate that the literal may or may not be negated.

$$\begin{aligned}
 &[\neg]p(\overline{t_{11}}) \vee \dots \vee [\neg]p(\overline{t_{1n_1}}) \\
 &\quad \vdots \\
 &[\neg]p(\overline{t_{m1}}) \vee \dots \vee [\neg]p(\overline{t_{mn_m}})
 \end{aligned}$$

Suppose we want to construct a definition for $\text{poss}_{p(\overline{x})}(\overline{x})$, i.e. a sentence $\phi(\overline{x})$ with free variables \overline{x} such that $\phi(\overline{t})$ is true if and only if $p(\overline{t})$ is consistent with the clauses above. First notice that this set of disjunctions is satisfiable if

and only if we can select one literal from each disjunction so that a literal and its negation are not both selected. $p(\bar{t})$ is satisfiable and therefore $poss_{p(\bar{t})}(\bar{t})$ is true if and only if there is a satisfying selection of literals that does not include $\neg p(\bar{t})$.

To construct a definition for $poss_{p(x)}(x)$, we take advantage of these observations by writing logical sentences that check whether there is a satisfying selection of literals that does not include $\neg p(\bar{t})$. Because that check requires analyzing the structure of the disjunctions at the metalevel, we reify the structure so that analysis can take place within the logic. For each disjunction $[\neg]p(\bar{t}_{i1}) \vee \dots \vee [\neg]p(\bar{t}_{ini})$ we invent a new object constant, conveniently spelled $\pm \bar{t}_{i1} \dots \pm \bar{t}_{ini}$ where the \pm means that a $+$ is used if the literal is positive and a $-$ is used if the literal is negative. In addition, we impose an ordering on the disjunctions. The new binary predicate d is true of $\langle k, c \rangle$ if and only if c is the object constant representing the k th disjunction.

$$d(x, y) \Leftrightarrow \left(\begin{array}{c} (x = 1 \wedge y = \pm \bar{t}_{11} \dots \pm \bar{t}_{1n_1}) \\ \vee \dots \vee \\ (x = m \wedge y = \pm \bar{t}_{m1} \dots \pm \bar{t}_{mn_m}) \end{array} \right)$$

This reification technique encodes incomplete information within a complete definition. To extract that incomplete information, we use $part$, which also has a complete definition. $part$ is a ternary predicate such that $part(\pm \bar{t}_{ij} \dots \pm \bar{t}_{ini}, x, y)$ is true whenever y is one of the \bar{t}_{ij} and x is its corresponding sign, i.e. $+$ or $-$.

Using d and $part$, we can construct a definition for all consistent selections of literals from the m disjunctions, where one literal is selected per disjunction. The definition of sat_1 is naturally recursive. Suppose we had a selection of literals from the last $m - 1$ disjunctions. Then to select a literal from the remaining disjunction requires simply finding a literal that is consistent with the selection so far. The definition for selection from the last $m - i$ disjunctions depends on a definition for selecting from the last $m - (i + 1)$ disjunctions. Often such a definition is written recursively, but because at the time this definition is written, the number of disjunctions is known, we can instead unroll the recursion and write m definitions. This unrolling is important because when the definitions are nonrecursive, they comprise a complete theory, which is the goal of this reformulation.

$$\begin{array}{l} sat_1(x_1, \bar{y}_1, \dots, x_m, \bar{y}_m) \Leftrightarrow \left(\begin{array}{l} \exists z. (d(1, z) \wedge part(z, x_1, \bar{y}_1)) \wedge \\ sat_2(x_2, \bar{y}_2, \dots, x_m, \bar{y}_m) \wedge \\ (\bar{y}_1 = \bar{y}_2 \Rightarrow x_1 = x_2) \wedge \\ (\bar{y}_1 = \bar{y}_3 \Rightarrow x_1 = x_3) \\ \wedge \dots \wedge \\ (\bar{y}_1 = \bar{y}_m \Rightarrow x_1 = x_m) \end{array} \right) \\ \vdots \\ sat_m(x_m, \bar{y}_m) \Leftrightarrow \exists z. (d(m, z) \wedge part(z, x_m, \bar{y}_m)) \end{array}$$

Finally we need a definition for $poss_{p(\bar{x})}(\bar{x})$ that is true whenever there is some sat_1 where none of the arguments are \bar{x} with sign $-$.

$$poss_{p(\bar{x})}(\bar{x}) \Leftrightarrow \exists x_1, \bar{y}_1, \dots, x_m, \bar{y}_m. \left(\begin{array}{l} sat_1(x_1, \bar{y}_1, \dots, x_m, \bar{y}_m) \wedge \\ \neg(x_1 = - \wedge \bar{y}_1 = \bar{x}) \\ \wedge \dots \wedge \\ \neg(x_m = - \wedge \bar{y}_m = \bar{x}) \end{array} \right)$$

A constraint that says p may be true of no more than n tuples can be handled by checking whether a satisfying selection of literals contains more than n positive literals or not.

To illustrate, we use the Wumpus World, which includes ground clauses and size constraints for *gold* and *wumpus*. Consider just the *gold* sentences.

$$\begin{aligned} gold(a) \vee gold(f) \vee gold(i) \\ gold(x) \wedge gold(y) \Rightarrow x = y \end{aligned}$$

Because there is a single ground disjunction, the definitions for d and $part$ are particularly simple.

$$\begin{aligned} d(x, y) &\Leftrightarrow (x = 1 \wedge y = +a + f + i) \\ part(x, y, z) &\Leftrightarrow \left(\begin{array}{l} (x = +a + f + i \wedge y = + \wedge z = a) \vee \\ (x = +a + f + i \wedge y = + \wedge z = f) \vee \\ (x = +a + f + i \wedge y = + \wedge z = i) \end{array} \right) \end{aligned}$$

Because there is one ground clause, there is just one sat predicate.

$$sat_1(x_1, y_1) \Leftrightarrow \exists z. (d(1, z) \wedge part(z, x_1, y_1))$$

The definition for $poss_{gold(x)}(x)$ is shown here, and takes into account the fact that there can be only one value for *gold*.

$$poss_{gold(x)}(x) \Leftrightarrow \exists x_1 y_1. \left(\begin{array}{l} sat_1(x_1, y_1) \wedge \\ \neg(x_1 = - \wedge y_1 = x) \wedge \\ \neg(x_1 = + \wedge y_1 \neq x) \end{array} \right)$$

In general, the definitions prescribed in this section are fairly efficient to compute. The definitions for d and $part$ cost time linear in the size of the clause set. The sat definitions altogether cost time quadratic in the number of clauses, and the cost of computing $poss_{p(\bar{x})}(x)$ is linear in the number of clauses.

This $poss$ definition construction can be generalized to handle ground clauses that constrain multiple predicates by including information in the new object constants about not only the sign of the tuple but also the predicate it belongs to, and the same complexity analysis holds.

6 Conclusion and Future Work

Extensional Reasoning is a promising new approach to automated theorem proving that leverages one of the most successful applications of logic today: the

relational database. The two key insights discussed in this paper are (1) how *poss* can be used to apply Extensional Reasoning to entailment queries about incomplete theories and (2) how *poss* definitions can be constructed efficiently by creating new object constants that represent syntactic features of an axiom set. By reformulating a theory into a complete theory about *poss*, and rewriting the query in terms of *poss*, ER techniques developed for complete theories can sometimes be leveraged to answer entailment queries orders of magnitude more quickly than traditional techniques.

The first and most obvious extension to the work presented here is an expansion of the class of theories that can be completed using *poss*. Ground clauses form a good starting point but are less compelling than we would like. We have experimented with computing *poss* definitions via abduction, but because our abduction algorithms relied on traditional proof techniques, run-times were unpredictable, and for some problems a great deal of time was spent on the reformulation step. It is unclear at this point whether there are better techniques for computing *poss* definitions in general.

The laws for distributing *poss* across logical connectives are relatively complete, but further conditions that identify independent sentences may turn out to be important. Such conditions would enlarge the class of cases for which *poss* distributes over conjunction, thereby breaking the *poss* definition computation up into smaller pieces.

Finally, there is the possibility that we could turn this work upside down. Instead of viewing the task as using databases to speed-up theorem proving, we could also view it as expanding traditional database techniques to allow for disjunctive data and more flexible query languages. (We are currently analyzing how this work relates to the work on incomplete databases, e.g. [56].)

References

1. Hinrichs, T.L., Genesereth, M.R.: Extensional reasoning. preparation (2007). <http://logic.stanford.edu/~thinrich/papers/hinrichs2007extensional.pdf>
2. Reiter, R.: On closed world data bases. Logic and Databases (1978)
3. Lloyd, J.: Foundations of Logic Programming. Springer, Heidelberg (1984)
4. McCarthy, J.: Circumscription - a form of non-monotonic reasoning. Artificial Intelligence (1980)
5. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, London (1995)
6. Zimanyi, E., Pirotte, A.: Imperfect knowledge in databases. In: Motro, A., Smets, P. (eds.) Uncertainty Management in Information Systems: From Needs to Solutions, pp. 35–87. Kluwer Academic Publishers, Dordrecht (1997)

An Abstract Theory and Ontology of Motion Based on the Regions Connection Calculus

Zina M. Ibrahim and Ahmed Y. Tawfik

University of Windsor, 401 Sunset Avenue,
Windsor, Ontario N9B 3P4, Canada
{ibrahim,atawfik}@uwindsor.ca

Abstract. In this paper, we present a framework abstracting motion by creating a qualitative representation of classes describing motion, and use the continuity constraints implicitly embedded in the semantics of these classes to create a framework that enables plausible reasoning about them. In particular, we propose a topology-based calculus of motion whose primitive is a *motion class*. We subsequently construct a set of primitive motion classes that exhaustively describes the change in topology between two moving objects, and show how compound motion classes are formed from these primitive motion classes using continuity constraints we make explicit. We use composition tables to define queries in the spatio-temporal domain and enable the extension of the classes to reason about the change in topology among three objects as they move.

1 Introduction

Qualitative formalisms have been accepted as suitable methods for high-level reasoning about spatial and spatio-temporal knowledge [2]. This is due to the fact that the epistemic nature of spatio-temporal information renders it vague, highly-dynamic or unknown, making it difficult to obtain the numerical values necessary for a quantitative representation. Because of this, numerical approaches to reasoning about the spatio-temporal domain are limited [2], and qualitative formalisms that provide an abstraction of such domains have prevailed [3].

The general principle of qualitative spatio-temporal representations is to capture one or more aspects of space, e.g. topology [1] [4] [5] [6], orientation [7] or distance [9] and formulate a set of relations that represents the possible interactions among two spatial objects with respect to the chosen aspect of space [3]. The resulting set of relations is must satisfy the property of being *Jointly Exhaustive and Pairwise Disjoint* (JEPD), which means that only one relation may hold between any two spatial objects at a time, and that together, the relations make up all forms of interaction among spatial objects [3]. Reasoning is performed by means of a composition table [3], which uses known relations between certain objects to predict unknown relations that may hold between other objects.

Because of the relational nature of the qualitative representations of space, opposed to the absolute Newtonian numerical approaches, it seemed natural to use to describe the elements of this calculi, and furthermore, use the elements of the calculi provide natural language descriptions of motion with respect to the calculi. For example [11] uses the topology-based qualitative theory of space of [1] to construct a set of six motion classes consisting of *leave*, *reach*, *cross*, *hit*, *internal* and *external*. [8] does the same but embeds vagueness in the representation of the regions under study to come up with a set of motion classes specific to vague regions.

What is common among the above sets of classes of motion is that their formulations were rather ad-hoc. More specifically, because they were directed towards capturing some semantics of motion and not to use it as a construct for a calculus of change, they did not necessitate that the set of motion classes formulated possess the properties necessary for a calculus. More specifically, they are all non-exhaustive, i.e. they do not represent all the forms that motion can take, which renders them unusable for a calculus of motion.

In this paper, we present a qualitative calculus of motion. The paper aims at formulating an abstract representation at motion by using a qualitative representation of space to create its primitive, a *motion class*, presents a high-level view of motion by ridding its of its quantitative aspects. We begin by defining the necessary conditions for a set of classes to become the primitive of such calculus, then we present a methodology for deriving the elements of the set of classes constituting the calculus from a jointly-exhaustive and pairwise-disjoint set of spatial relations. We follow by introducing reasoning mechanisms that enables the prediction of the possible topologies between moving objects other than the ones we have knowledge about.

Although the formalism we present is a general one and can be used to derive motion classes from a qualitative spatial calculus that represents any aspect of space, for readability purposes, we choose introduce it by deriving its constructs from the well known spatial theory of the *Region Connection Calculus* (RCC8) [12]. The result is an abstract theory of motion where topology determines the spatial interactions among the objects under study.

The paper is organized as follows. Section 2 reviews the RCC8 calculus concentrating on the continuity properties it possesses, and how the RCC8 conceptual neighborhood captures spatio-temporal relationships. Section 3 presents the spatio-temporal representation based on motion classes and Section 4 shows how to combine two motion classes to form a compound motion. Section 5 constructs reasoning mechanisms through the use of composition tables to formulate spatio-temporal queries about motion. Finally Section 6 present a summary, some conclusions and future directions.

2 The Region Connection Calculus: RCC8 Set

Although many qualitative theories for dealing with space exist in the literature, the RCC8 calculus [12] has attracted more attention within the QSR community

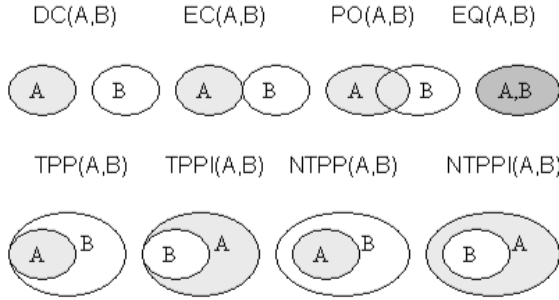


Fig. 1. The RCC8 Relations

than others. RCC8 uses the notion of 'connectedness' to formulate a set of JEPD topological relations that may hold between two regions at any time. For any two regions x and y , the topological relation that may exist between x and y is represented by $r_i(x,y) \in \text{RCC8}$ and can be one of the following: $\text{DC}(x,y)$ (x is disconnected from y), $\text{EC}(x,y)$ (x is externally connected to y), $\text{PO}(x,y)$ (x partially overlaps y), $\text{EQ}(x,y)$ (x is equal to y), $\text{TPP}(x,y)$ (x is a tangential proper part of y) and its inverse $\text{TPPI}(x,y)$ (y is a tangential proper part of x), $\text{NTPP}(x,y)$ (x is a non-tangential proper part of y) and its inverse $\text{NTPPI}(x,y)$ (y is a non-tangential proper part of x). Figure 1 shows the complete set.

Reasoning is performed by means of a composition table [12], which generally takes the form of answering the following query: for three spatial regions x , y and z , and RCC8 relations rcc_i and rcc_j , if $\text{rcc}_i(x,y)$ and $\text{rcc}_j(y,z)$ hold, what are the possible RCC8 relations that may hold between x and z ?

In addition to its high expressive power and flexibility, the RCC8 relations possess an inherit continuity property in which each relation can be mapped to another from the set via a continuous transition. This property has been modeled by what is known as the RCC8's conceptual neighborhood graph (CNG) [12] shown in figure 2.

Figure 2 shows the continuity of the process of transforming one RCC8 relation to another. Transforming some $\text{rcc}_i \in \text{RCC8}$ to some $\text{rcc}_j \in \text{RCC8}$ can be done via one direct transition (e.g. DC to PO), making rcc_i and rcc_j immediate conceptual neighbors, or indirectly (e.g. PO and DC). In the indirect case, there must be some sequence S of RCC8 relations that connect rcc_i and rcc_j . The fact that S cannot be empty ensures the continuity of the transitions among members of RCC8.

Because of the continuity properties explained above, the RCC8 relations can be interpreted to model the topological relations between the spatio-temporal evolutions of spatial regions in addition to topological relations between static spatial regions. An example is shown in figure 3 where the relation PO (partial overlap) between two regions x and y is shown to hold during the temporal

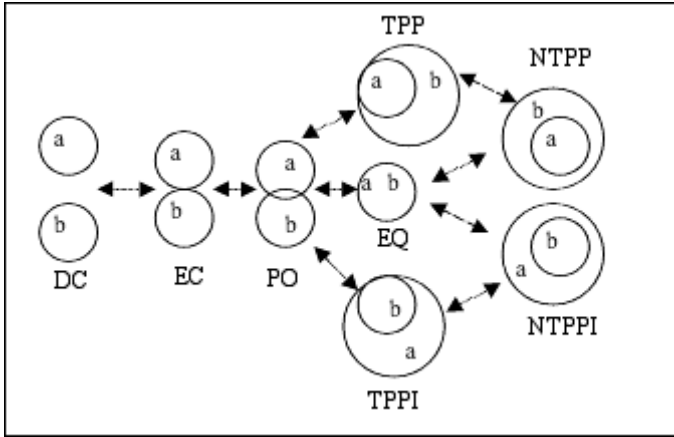


Fig. 2. The RCC8 Conceptual Neighborhood

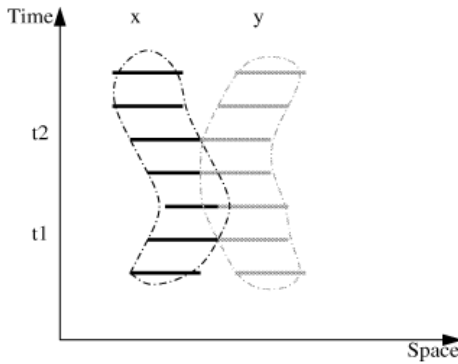


Fig. 3. Spatio-temporal Interpretation of PO

interval $[t_1, t_2]$. In the figure, the horizontal and vertical axes represent the evolution of space and time respectively. Hence, the figure shows not the static regions x and y , but the evolution of their spatial extents through time. Hence, the relation $PO(x,y)$ holds when the evolution of region x overlaps that of region y during some interval, e.g. $[t_1, t_2]$.

Because of its spatio-temporal nature, it is possible to construct from the RCC8 set a set of motion class descriptions. In [11], six mutually exclusive classes have been formulated. They are $leave(z, x, y)$, $reach(z, x, y)$, $hit(z, x, y)$, $cross(z, x, y)$, $internal(z, x, y)$ and $external(z, x, y)$ representing in each case the motion class that holds between two spatio-temporal moving regions x and y during interval z .

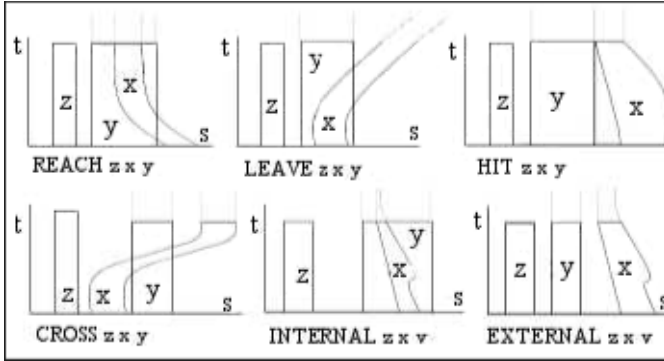


Fig. 4. Muller’s Motion Classes

Figure 4 shows the six classes and is read as follows. The horizontal axis represents the spatial extent of the region and the vertical axis represents the temporal evolution. Hence, the beginning of interval z is located in its lowest t value and its end is located at its highest t value. The spatial extents of regions x and y evolve as time increases forming a motion class. We do not give the formal definition of these classes. The interested reader can refer to [11].

3 A Qualitative Set of Motion Classes

In this section, we present the building blocks for a calculus which abstracts the motion of one spatial region with respect to another by describing it via a set of classes which it can take. More specifically, given two regions x and y moving during the interval $[t, t + \Delta]$, we study the change in the topological relations that hold between the two regions at the beginning and end of the interval, and use them to categorize motion by constructing a set of classes that it may belong to.

The topology at t and $t + \Delta$ is described by the RCC8 relations that hold between x and y at t and $t + \Delta$ respectively. Because RCC8 has 8 relations, all adhering to a continuous neighborhood structure, there exists a total of $|RCC8|^2 = 64$ possible forms of motion. In this section, we construct the set MC, which is an exhaustive set of motion classes, determined by the 64 possibilities.

Hence, our calculus consists of a topological spatial base (RCC8) and a set of motion classes derived from it (MC). Before we introduce MC, we reformulate the properties of the RCC8 set to make it compatible with our calculus.

3.1 The Spatial Base

Consists of the set $RCC8 = \{DC, EC, PO, EQ, TPP, TPPI, NTPP, NTPPI\}$. In this paper, we adopt the notation $rcc_{[t, t + \Delta]}(x, y)$ to denote the following:

1. RCC8 relation rcc holds between two regions x and y during the entire duration of the interval $[t, t + \Delta]$.

2. It is not prohibited for rcc to hold between x and y before $[t, t + \Delta]$ or to persist after it. The knowledge of rcc before or after $[t, t + \Delta]$, however, is not readily available.

Elements of the RCC8 set possess the following properties:

1. They are Jointly Exhaustive: Any topological relation that may hold between two regions of space x and y during the interval $[t, t + \Delta]$ must belong to RCC8.

$$\forall rcc, rcc_{[t, t + \Delta]}(x, y) \rightarrow rcc \in \text{RCC8}$$

2. They are Pairwise Disjoint: No two RCC8 relations can hold between two spatial regions x and y at the same time.

$$\forall rcc_1, rcc_2 \in \text{RCC8} : (rcc_1_{[t, t + \Delta]}(x, y) \wedge rcc_2_{[t, t + \Delta]}(x, y)) \rightarrow rcc_1 = rcc_2$$

3. They Adhere to a Conceptual Neighborhood structure:

$$\forall rcc_1 \in \text{RCC8} , \exists rcc_2 \in \text{RCC8} : rcc_1 \neq rcc_2 \wedge \mu(rcc_1, rcc_2) = 1$$

Where $\mu(rcc_1, rcc_2)$ is the number of arcs separating rcc_1 and rcc_2 in RCC8's conceptual neighborhood graph.

3.2 The Set MC of Motion Classes

As explained above, the primitives of our calculus, the elements of $\text{MC} = \{mc_1, \dots, mc_n\}$ are characterized by the RCC8 relations that hold at the beginning and end of the interval $[t, t + \Delta]$ during which they hold.

For this purpose, we define the predicates **starts** $(mc_i, [t, t + \Delta], x, y)$ and **ends** $(mc_i, [t, t + \Delta], x, y)$, where **starts** $(mc_i, [t, t + \Delta], x, y)$ returns a subset of RCC8 corresponding to the relations that can hold between spatio-temporal objects x and y at the beginning of the interval $[t, t + \Delta]$ during which the class mc_i correctly describes the change taking place (i.e. at time t), while **ends** $(mc_i, [t, t + \Delta], x, y)$ gives the set of RCC8 relations that hold between spatio-temporal objects x and y at the end of the interval $[t, t + \Delta]$ during which the class mc_i holds (i.e. at $t + \Delta$). They are both given in definitions 1 and 2 below:

Definition 1. **starts** $(mc_i, [t, t + \Delta], x, y) = \{rcc \mid rcc \in \text{RCC8} \wedge rcc_t(x, y)\}$.

Definition 2. **ends** $(mc_i, [t, t + \Delta], x, y) = \{rcc \mid rcc \in \text{RCC8} \wedge rcc_{(t + \Delta)}(x, y)\}$.

It is important to note before continuing, that the interval $[t, t + \Delta]$ is defined as the minimal interval required to inflict spatial change. In other words, for some motion class mc , if **starts** $(mc, [t, t + \Delta], x, y) = \{\text{DC}\}$ and **ends** $(mc, [t, t + \Delta], x, y) = \{\text{PO}\}$, then the only relations allowed during the interval are those relations that lie between DC and PO in RCC8's conceptual neighborhood graph, which are the relations necessary for the change to occur. Also, these relations should only hold in the order that allows for mc to hold during the interval.

3.3 Constructing the Set MC

The construction of MC is initiated by the fact that the elements of the set $RCC8^2$ do not uniquely describe $|RCC8|^2=64$ classes of motion. For example, any motion that starts with the regions being in any configuration except disconnected and ends with the regions being disconnected represents that one region left the other. Hence, there exists a one-to-many mapping between MC and $RCC8^2$. This mapping is defined below.

We define the set $j = \{j_1, \dots, j_n\}$, where for each motion class $mc_i \in MC$, there exists $j_i = \{(rcc_1, rcc_2), \dots (rcc_r, rcc_s)\} \in j$ representing the subset of $RCC8^2$ that describes all the possible pairs of spatial relations that hold at the beginning and end of the interval during which the class mc_i holds (e.g. $rcc_1 \in \mathbf{starts}(mc_i, [t, t + \Delta], x, y)$ and $rcc_2 \in \mathbf{ends}(mc_i, [t, t + \Delta], x, y)$ for two regions x and y moving during interval $[t, t + \Delta]$). Unlike the case with $RCC8$, there exists a one-to-one correspondence between j and MC. Property 3.3.1 formalizes the above concept.

Property 3.3.1. $\forall mc_i \in MC, \exists j_i = \{(rcc_1, rcc_2), \dots, (rcc_r, rcc_s)\} \in j: \forall (rcc_l, rcc_m) \in j_i, rcc_l \in \mathbf{starts}(mc_i) \wedge rcc_m \in \mathbf{ends}(mc_i)$.

As a result, nine motion classes have been identified and are given in table 1. The motion classes are LEAVE, REACH, HIT, SPLIT, PERIPHERAL, EXPAND, SHRINK, INTERNAL and EXTERNAL. In the table, the rows correspond to the $RCC8$ relation which belongs to the set $\mathbf{starts}(mc_i, [t, t + \Delta], x, y)$ while the column corresponds to the $RCC8$ relation which belongs to the set $\mathbf{ends}(mc_i, [t, t + \Delta], x, y)$. Each intersection of a row and a column presents the resulting motion class when the corresponding $RCC8$ relations at the beginning and end of the interval hold.

Table 1. The Set MC of Motion Classes

		ENDS								
		DC	EC	PO	TPP	NTPP	EQ	TPPi	NTPPi	
STARTS	DC	EXTERNAL	HIT	REACH						
	EC	SPLIT	PERIPHERAL	REACH						
	PO			LEAVE OR REACH						
	TPP					INTERNAL	EXPAND			
	NTPP							EXPAND		
	EQ			LEAVE				INTERNAL		
	TPPi					SHRINK		INTERNAL		
	NTPPi							INTERNAL		

3.4 Properties of MC

In order for the elements of the set MC to be useful in being primitives in a calculus of motion classes, they must, along with the spatial base $RCC8$, exhaustively describe all possible forms of motion between two regions x and y during interval $[t, t + \Delta]$, and possess continuity properties by having a conceptual neighborhood structure. Below we outline these properties and show that elements of MC indeed possess them.

Completeness

Claim 3.4.1 *MC exhausts all forms of spatio-temporal change during a given interval. In other words, MC covers RCC8².*

Justification:

By construction, the set \mathcal{J} covers RCC8². As a result

$$\bigcup_{j_i \in \mathcal{J}} j_i = \text{RCC8}^2$$

Since there exists a one-to-one correspondence between \mathcal{J} and MC, MC in turn covers RCC8².

Continuity

Claim 3.4.2 *The set of motion classes MC adheres to a conceptual neighborhood structure. In other words, for every class $mc_i \in \text{MC}$, there must exist another class $mc_j \in \text{MC}$ which represents the linguistic continuation of mc_i .*

Justification:

1. Existence:

Dictated by definitions 1 and 2, for all motion classes mc_i and $mc_j \in \text{MC}$, $\text{starts}(mc_i)$, $\text{ends}(mc_i)$, $\text{starts}(mc_j)$, $\text{ends}(mc_j) \in \text{RCC8}$.

However, because the RCC8 adheres to a conceptual neighborhood structure, for every RCC8 relation rcc_i , there exists another RCC8 relation rcc_j such that rcc_i and rcc_j are conceptual neighbors in RCC8's conceptual neighborhood graph.

As a result, mc_i and mc_j are conceptual neighbors if $\text{ends}(mc_i)$ and $\text{starts}(mc_j)$ are conceptual neighbors in RCC8's conceptual neighborhood graph. Now, since members of MC cover RCC8², the pair $(\text{ends}(mc_i), \text{starts}(mc_j))$ must belong to RCC8². Hence, the motion class mc_j that represents the continuous extension of mc_i must exist.

2. Necessity:

Assume that for three motion classes mc_i, mc_j and $mc_k \in \text{MC}$, mc_i and mc_j are conceptual neighbors, mc_j and mc_k are conceptual neighbors, while mc_i and mc_k are not conceptual neighbors. Also, assume that mc_i has evolved to mc_k without going through the motion class mc_j .

This means that $\text{ends}(mc_i)$ and $\text{starts}(mc_k)$ must be conceptual neighbors in the RCC8's conceptual neighborhood graph. However, this forms a contradiction, because if this is true, then mc_i and mc_k would have been conceptual neighbors.

Therefore, members of MC adhere to constraints specified by their conceptual neighborhood but forming only continuous transitions.

Having justified the existence of MC's conceptual neighborhood graph, we now give a formal definition of two motion classes being conceptual neighbors. The definition is given in 3 and is read as follows: motion classes mc_i holding during

interval $[t, t + \Delta]$ and mc_j holding during the interval which immediately follows $[t, t + \Delta]$ (i.e. interval $[t + \Delta, t + \Delta + \delta]$ in the definition) are conceptual neighbors if one of the RCC8 relations that may hold at the end of mc_i is a conceptual neighbor of one of the RCC8 relations that may hold at the beginning of mc_j in RCC8's conceptual neighborhood graph.

Definition 3. *conc_neigh* ($mc_i [t, t + \Delta] x, y$), ($mc_j [t + \Delta, t + \Delta + \delta] x, y$) if $\exists s_1 = \mathbf{starts}(mc_j [t + \Delta, t + \Delta + \delta] x, y)$, $s_2 = \mathbf{ends}(mc_i [t, t + \Delta] x, y) \wedge \exists r_1 \in s_1 \wedge r_2 \in s_2: \theta \leq \mu(r_1, r_2) \leq 1$.

We show the conceptual neighborhoods of the set MC in table 2 below. We resort to the table notation instead of the well-known conceptual neighborhood graph for visibility purposes. The table should be read as follows. The rows represent

Table 2. MC's Conceptual Neighbors

Motion Class	Conceptual Neighbors
LEAVE	All
REACH	All but Hit and External
HIT	Leave, Reach, Split, Peripheral, External
SPLIT	Reach, Hit, Peripheral, External
PERIPHERAL	Leave, Reach, Hit, Split, External
EXPAND	Leave, Reach, Shrink, Internal
SHRINK	Leave, Reach, Expand, Internal
INTERNAL	Leave, Reach, Expand, Shrink
EXTERNAL	Reach, Hit, Split, Peripheral

4 Compound Motion Classes

As the name implies, a compound motion consists of two primitive motions. However, to be able to compose two primitive motions, the following conditions should be satisfied:

1. Motion continuity dictates that the spatial relation that holds at the end of the first motion also holds for the start of the second motion. Formally, to compose mc_i and mc_j , there should exist $rcc_i \in \text{RCC8}$ such that $rcc_i \in \mathbf{ends}(mc_i)$ and $rcc_i \in \mathbf{starts}(mc_j)$. For example, it is possible to form a compound motion from HIT and SPLIT but it is not possible to combine HIT and INTERNAL.
2. To eliminate redundancy, the two motions forming the compound motion should not be subsumed by a primitive motion. Formally, this condition implies that for the sequence of RCC8 relationships visited by the compound motion composed of mc_i and mc_j does not correspond to the sequence generated from any single motion mc_k . For example, REACH subsumes the composition of HIT and REACH.

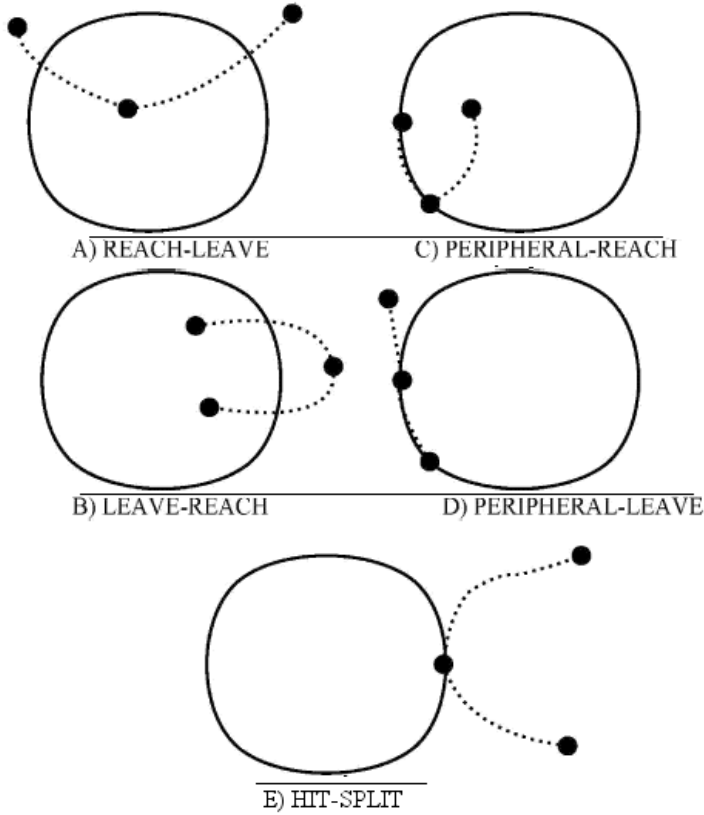


Fig. 5. Compound Motion Classes

Figure 5 illustrates five compound motion classes satisfying the two above conditions. These compound motions include REACH-LEAVE (or CROSS), PERIPHERAL -REACH, LEAVE-REACH, PERIPHERAL-LEAVE, and HIT- SPLIT (or TOUCH). In the figure, the dotted lines represent the trajectory of the moving region with respect to another region. The start and end of each constituent motion are marked with dark bullets.

5 Queries About Motion Classes

Now that we have established the set of motion classes, and we have shown that they are exhaustive to the set $RCC8^2$, and that they adhere to a conceptual neighborhood structure, we can now use composition tables to reason about the set MC^2 to obtain spatio-temporal information regarding three objects in motion as illustrated in section 5.1 below.

Table 3. Composition Table Representing Spatio-temporal Queries

Motion	LEAVE	REACH	HIT	SPLIT	PERIPHERAL
LEAVE	Ext. Leave Int.	Ext.	Ext.	Ext. Hit Reach	Ext. Split
REACH	Ext.	Reach Int.	Hit Periph. Ext.	Ext.	Hit Ext.
HIT	Ext.	Int. Periph. Hit	Int.	Ext. Leave	Ext. Reach
SPLIT	Leave Int.	Ext.	Ext.	Int. Ext.	Leave Ext.
PERIPHERAL	Ext. Leave	Hit Reach	Ext. Reach	Ext. Leave	Ext. Int.
EXPAND	Ext. Leave Exp.	Reach	Reach	Hit Reach Ext.	Reach
SHRINK	Leave	Leave, Reach Ext.	Ext.	Leave Ext.	Leave
INTERNAL	Leave Ext. Split	Reach Ext. Hit	Ext.	Ext.	Ext.
EXTERNAL	All but Shrink	All but Expand	All but Expand	All but Shrink	All but Exp. & Shrink

Table 4. Composition Table Representing Spatio-temporal Queries

Motion	INTERNAL	EXTERNAL	EXPAND	SHRINK
LEAVE	Int. Leave	Reach Ext. Hit	Leave Int.	Leave Peri. Split Int. Ext.
REACH	Reach Int.	Leave Split Ext.	Reach	Reach
HIT	Reach Int.	Leave Ext. Split	External	Reach
SPLIT	Int. Leave	Reach Hit Ext.	Leave	Ext.
PERIPHERAL	Int.	Leave Reach Hit Split Ext.	Leave	Reach
EXPAND	Int. Leave	Ext. Reach Hit	Expand	Int. Reach Expand
SHRINK	Shrink Int.	Ext. Leave Split	Int. Leave	Shrink
INTERNAL	Int.	Ext.	Leave Int. Expand	Reach Int. Shrink
EXTERNAL	Ext. Int. Leave Reach Hit Split Shrink	All	Ext. Leave Split	Ext. Reach Hit

5.1 Queries About Spatio-temporal Knowledge: Composition Tables

During interval $[t, t + \Delta]$, given three objects x , y and z undergoing motion, where $mc_i(x, y)$ holds and $mc_j(y, z)$ holds, what are the possibilities for the motion class between x and z ?

Tables 3 and 4 answer this question for all members of MC. In the table, the rows represent the possibilities for the motion class $mc_i(x, y)$ while the columns represent $mc_j(y, z)$. The intersection of a row and a column represent the possible motion classes $mc_k(x, z)$ during the interval $[t, t + \Delta]$.

6 Conclusions and Future Research

Continuity of spatio-temporal evolutions as encoded in the conceptual neighborhood graph is an important property that can be exploited to represent and reason about motion. This paper has proposed a framework for exploiting continuity in the regions connection calculus RCC8 to define an abstract framework that represents motion based on classes. The framework distinguishes between a set of simple or primitive motion classes that can be combined to form compound motion. In addition, the paper shows how to use the set of motion class to answer queries in the spatio-temporal domain based on composition tables. Extending the proposed framework to handle probabilistic and uncertain spatio-temporal evolutions; and exploring applications for the proposed framework are two directions of future investigation.

Acknowledgements

The authors would like to acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) through the discovery grants program and the support of the Ontario Graduate Scholarship (OGS) program.

References

1. Asher, N., Vieu, L.: Toward a Geometry for Common Sense: A Semantics and a Complete Axiomatization for Mereotopology. In: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence. pp. 846–852 (1995)
2. Bailey-Kellog, C., Zhao, F.: Qualitative Spatial Reasoning: Extracting and Reasoning with Spatial Aggregates. *AI Magazine* 24(4), 47–60 (2003)
3. Cohn, A., Hazarika, S.: Qualitative Spatial Representation and Reasoning: An Overview. *Fundamenta Informatica* 46(1-2), 2–32 (2001)
4. Randell, D., Cui, Z., Cohn, A.: A Spatial Logic based on Regions and Connection. In: Proceedings of the International Conference on Knowledge Representation and Reasoning, pp. 165–176 (1992)
5. Egenhofer, M., Frenzo, R.: On the Equivalence of Topological Relations. *International Journal of Geographical Information Systems* 9(2), 133–152 (1995)

6. Guesgen, H.: When Regions Start to Move. Proceedings of the Florida Conference on Artificial Intelligence, pp. 465–469 (2003)
7. Hernandez, D.: Qualitative Representation of Spatial Knowledge. LNCS, vol. 804, Springer, Heidelberg (1994)
8. Ibrahim, Z., Tawfik, A.: Spatio-temporal Reasoning for Vague Regions. In: Proceedings of the Canadian Conference on Artificial Intelligence, pp. 308–320 (2004)
9. Köhler, C.: The Occlusion Calculus. In: Cognitive Vision Workshop (2002)
10. Muller, P.: Topological spatiotemporal reasoning and representation. Computational Intelligence 18(3), 420–450 (2002)
11. Muller, P.: A Qualitative Theory of Motion Based on Spatio-temporal Primitives. In: Proceedings of the International Conference on Knowledge Representation and Reasoning (1998)
12. Randell, D., Cui, Z., Cohn, A.: A Spatial Logic based on Regions and Connection. In: Proceedings of the International Conference on Knowledge Representation and Reasoning, pp. 165–176 (1992)

Computing and Using Lower and Upper Bounds for Action Elimination in MDP Planning

Ugur Kuter¹ and Jiaqiao Hu²

¹ University of Maryland Institute for Advanced Computer Studies,
University of Maryland at College Park,
College Park, MD 20742, USA
ukuter@cs.umd.edu

² Department of Applied Mathematics and Statistics,
State University of New York at Stony Brook,
Stony Brook, NY 11794, USA
jqhu@ams.sunysb.edu

Abstract. We describe a way to improve the performance of MDP planners by modifying them to use lower and upper bounds to eliminate non-optimal actions during their search. First, we discuss a particular state-abstraction formulation of MDP planning problems and how to use that formulation to compute bounds on the Q-functions of those planning problems. Then, we describe how to incorporate those bounds into a large class of MDP planning algorithms to control their search during planning. We provide theorems establishing the correctness of this technique and an experimental evaluation to demonstrate its effectiveness. We incorporated our ideas into two MDP planners: the *Real Time Dynamic Programming* (RTDP) algorithm [1] and the *Adaptive Multi-stage* (AMS) sampling algorithm [2], taken respectively from automated planning and operations research communities. Our experiments on an Unmanned Aerial Vehicles (UAVs) path planning problem demonstrate that our action-elimination technique provides significant speed-ups in the performance of both RTDP and AMS.

1 Introduction

In planning under uncertainty, the planner's objective is to find a policy that optimizes some expected utility. Most approaches for finding such policies are based on Markov Decision Process (MDP) models. Despite their general applicability and mathematical soundness, the problem of generating optimal policies in MDPs is often computationally challenging due to the typical requirement of enumerating the entire state and/or action spaces. In order to address this issue, many general MDP-based solution methods have been developed: examples include *state abstraction and aggregation techniques* [3,4,5,6,7], *feature extraction methods* [8], *value function approximations* [9,10], *heuristic and greedy search* [11,11], and various *simulation-based techniques* [12,13,14,2].

The key idea throughout the above works is to reduce the size of the search space while preserving the correctness of the planning algorithms, and in doing

so, to generate solutions to MDP planning problems within reasonable running times. Despite these advances, MDP planning problems still pose a computational challenge due to the need to explore all or most of the state space.

This paper focuses on a way to improve the efficiency of planning on MDPs by computing lower and upper bounds on the Q-functions of MDP planning problems and using these bounds during planning to identify and eliminate actions that are guaranteed not to appear in optimal policies. We combine ideas from two different previous approaches: using state abstractions for MDPs to generate abstract versions of the original planning problems and using bounds for controlling the search of MDP planners. Our technique is a tightly-coupled framework based on a form of state abstraction that enables planning algorithms to identify and eliminate sub-optimal actions from MDPs and allows them to search the reduced MDP for effective planning.

In particular, our contributions are as follows:

- We discuss a particular state-abstraction formulation of MDP planning problems. In a large class of MDPs, the state space admits a factored representation [15], i.e., each state can be represented as a collection of state variables. In such planning problems where the state space is factored, often the cost of each action will not depend on all of the state variables, but only on some of them. Our technique exploits this property and generates equivalence classes of states (or equivalently, it generates a particular partition of the MDP) by examining the actions and the immediate cost of applying them.
- We describe how to compute lower and upper bounds on the Q-function of an MDP based on the equivalence classes over the states of that MDP. We also provide general conditions under which our technique can be incorporated into the existing MDP planning algorithms and can still guarantee to generate optimal solutions in the modified algorithms.
- We have applied our approach to the RTDP [1] and AMS [2] algorithms and performed several experiments with the modified planners. This study demonstrates that the modified algorithms generate solutions about two or three orders of magnitude faster than the original ones in a simplified version of an Unmanned Aerial Vehicles (UAVs) path planning domain.

2 Definitions and Notation

We consider MDP planning problems of the form $P = (S, A, T, \gamma, Pr, C, S_0, G)$, where S is a finite set of states and A is a finite set of actions. $T : S \times A \rightarrow 2^S$ is the state transition function, γ is a discount factor, Pr is a transition-probability function, C is a bounded one-stage cost function, S_0 is a set of initial states, and G is a set of goal states. The set of all actions applicable to a state s is $A_T(s) = \{a : T(s, a) \neq \emptyset\}$. If $a \in A_T(s)$, then $Pr(s, a, s')$ is the probability of going to the state s' if one applies the action a in s .

Although a policy is often defined to be a function $\pi : S \rightarrow A$, π need not always be total. If a state in S is unreachable from S_0 using π , then it can safely be omitted from the domain of π . Thus, we define a policy to be a partial

function from S into A (i.e., a function from some set $S_\pi \subseteq S$ into A) such that $S_0 \subseteq S_\pi$ and S_π is closed under π and A (i.e., if $s \in S_\pi$ and $s' \in T(s, \pi(s))$, then $s' \in S_\pi$).

Given a policy π , the *value function* $V^\pi(s)$ is the expected sum of the future discounted costs, i.e., $V^\pi(s) = E_\pi[\sum_{t=0}^{\infty} \gamma^t C(s_t, \pi(s_t)) \mid s_0 = s]$, where s_t is the state of the MDP at time t , and $E_\pi[\cdot]$ is understood with respect to the sample space induced by the transition probabilities.

An *optimal solution* is a policy π^* such that when executed in the initial state s_0 , π^* reaches a goal state in G with probability 1, and no other policy π' has both the same property and a lower expected cost. It is well-known [16] that the optimal value $V^{\pi^*}(s)$ for state s can be computed by solving

$$V^{\pi^*}(s) = \begin{cases} 0, & \text{if } s \in G \\ \min_{a \in A_T(s)} Q(s, a), & \text{otherwise,} \end{cases} \quad (1)$$

$$Q(s, a) = C(s, a) + \gamma \sum_{s' \in T(s, a)} Pr(s, a, s') V^{\pi^*}(s'). \quad (2)$$

Since we have defined policies to be partial functions from S into A , it follows that the domain of the value function V^{π^*} is S_{π^*} .

In this paper, we assume that the state space S of an MDP planning problem admits a *factored* representation [15]; i.e., S can be represented by using a finite set of state variables $X = \{x_1, x_2, \dots, x_n\}$, where each state variable $x_i \in X$ takes values from a finite domain $dom(x_i)$. Every state $s \in S$ is a set of variable assignments $\{x_1 = v_1, \dots, x_n = v_n\}$. Note that if s and s' are two states, then $s \cap s'$ is the set of all variable assignments $x_i = v_i$ that appear in both s and s' .

Our key assumption is that the cost $C(s, a)$ of applying an action a to a state s only depends on *some* of the state-variable assignments in s , not on all of them. There are many cases where this assumption holds. For example, if we are at a store and are trying to decide between two different items that both have the same price p , then we can safely assume that the price we pay for either of them will still be p regardless of whether the traffic light outside is red or green.

As another example, consider a simplified version of an Unmanned Aerial Vehicle (UAV) path planning application described in [17]. The planning domain is modeled as an $N \times N$ grid world. There are three kinds of entities: a UAV, a number of *objects* whose photos need to be collected by the UAV, and a number of *threats* that are hazardous for the UAV. Figure 1 shows a possible configuration of the entities in the UAV world. The UAV has five actions: North, South, East, West, and CollectPhoto. A threat location is a location such that there is a possibility that the UAV may get destroyed if it enters that location. In this problem, the immediate cost of applying an action in a grid location is determined by only some of the state-variables that describe the world. For example, in Figure 1, the immediate cost of performing a North action from the UAV position shown on the grid depends only on the level of the adjacent threat to the north of the UAV; it does not depend on the levels of the other threats.

Based on the above assumption, we formalize the notion of an equivalence class as follows. Given a state s and an action a , let $s^a \subseteq s$ be the set of all of

				O
	O			T
	T	T		
	U		T	O

Fig. 1. An instance of a 5 × 5 UAV domain. Above, “U” represents the UAV, “O” represents an object to be collected by the UAV, and “T” represents a threat location.

the state-variable assignments needed to determine $C(s, a)$. Let $[s^a]$ be the set of all states s' such that $s^a \subseteq s'$. Then for every state $s_i \in [s^a]$, $C(s_i, a) = C(s, a)$ and $[s_i^a] = [s^a]$. Thus, the set $S^a = \{[s^a] : s \in S\}$ is a partition, and for each s , $[s^a]$ is an equivalence class in that partition.

3 Computing Lower and Upper Bounds

Given an MDP planning problem P in a factored representation, we compute the set D of all possible equivalence classes, by examining the description of each action a of the input MDP and identifying each equivalence class $[s^a]$. Note that the state-variables that are relevant to the applicability of a in a state and the cost of a in that state are often given in the MDP planning problem description. For example, in problems represented as Probabilistic STRIPS operators [18], the relevant state variables appear in the preconditions and the effects of the planning operators. In problems represented as 2TBNs [19], the state variables can be extracted directly from the 2TBN.

Once we have the set D of the equivalence classes for P , we proceed with computing the lower and upper bounds on the Q function of P . Figure 2 shows our algorithm, called **Generate-LowerBounds (gLB)**, for generating the lower bounds on the Q function. Formally, **gLB** generates a positive real number $c(s, a)$:

$$c(s, a) = C(s, a) + \min_{s' \in T(s, a), a' \in A_T(s')} c(s', a'),$$

for every state $s \in [s^a]$ and action a applicable in those states, and $c(s, a) = 0, \forall a \in A_T(s)$ if $s \in G$.

The inputs of **gLB** are D, D^{S_0} , and D^G , where $D^{S_0} \subseteq D$ is the set of all equivalence classes that contain an initial state $s \in S_0$, and $D^G \subseteq D$ is the set of all equivalence classes that contains a goal state. The algorithm computes the lower bound defined above by doing successive backward breadth-first searches over the equivalence classes in D , starting from the goal D^G towards D^{S_0} .

In Figure 2, the set F denotes the set of equivalence classes that have been visited by the backward search. That is, F is a set of pairs of the form $([s^a], c)$

```

Procedure Generate-LowerBounds( $D, D^{S_0}, D^G$ )
 $F \leftarrow \emptyset$ 
repeat
   $F' \leftarrow F$ 
   $F \leftarrow F \cup \{([s^a], 0) \mid [s^a] \in D^G\}$ 
   $F \leftarrow \text{ComputeLowerBounds}(\text{Preimage}(F, D))$ 
until  $F = F'$ 
if  $D^{S_0} \subseteq \{[s^a] \mid ([s^a], c) \in F\}$  then return  $F$ 
return failure

```

Fig. 2. The Generate-LowerBounds (gLB) procedure

where $[s^a]$ is an equivalence class and c is the minimum cost of reaching a goal by applying a in the states of $[s^a]$ computed so far. Initially, F is the empty set.

At each iteration, gLB first takes F at the current iteration and adds the equivalence classes corresponding to the goal states, G , of the input MDP planning problem. Note that the lower bound of the costs associated with the goal states is 0, since we do not apply any action to a goal state. Then, gLB performs a `Preimage` computation, which is the core of the algorithm. Intuitively, `Preimage` computes the set of equivalence classes in D such that for each equivalence class $[s^a]$ in `Preimage`(F, D), there exists an equivalence class in F that is reachable by one or more of the effects of the action a .

Formally, `Preimage` is defined as follows:

$$\text{Preimage}(F, D) = \{([s^a], c' + C([s^a])) \mid [s^a] \in D, ([s_i^{a_i}], c') \in F, s_i \in T(s, a), \text{ and } a_i \in A\},$$

where D is the set of all equivalence classes computed for the input MDP planning problem, and A , T , and C are the set of all actions, the state-transition function, and the cost function of the input MDP. Note that the `Preimage` computation generates all possible cost-to-go values for an equivalence class $[s^a]$ given the set F at this iteration. Thus, as a next step, gLB computes the minimum of all the cost-to-go values obtained via `Preimage` computations and assigns this minimum cost-to-go value to $[s^a]$. In Figure 2, the `ComputeLowerBounds` subroutine is responsible for performing this task for all equivalence classes in the result of `Preimage`.¹

The backward search of gLB terminates when there are no other possible updates to F ; i.e., the procedure cannot generate any more equivalence classes and it cannot update the cost-to-go values associated with each equivalence class in F . In that case, the cost-to-go values specified in F are lower bounds on the cost-to-go values computed for each $[s^a] \in F$ from which an equivalence class that contains a goal state is reachable. That is, if $F = F'$ then the procedure

¹ Here, we presented `ComputeLowerBounds` as a separate subroutine for clarity; however, note that the computation of `ComputeLowerBounds` can also be embedded in the `Preimage` subroutine.

has generated a lower bound c for applying an action a in the states $[s^a]$ in the execution structure K , and there is no other $c' < c$ for $[s^a]$. In this case, **gLB** checks whether the lower bounds in the $Q(s, a)$ values for the states in the initial equivalence classes in D^{S_0} are computed. If so, the procedure returns the set F . Otherwise, it returns failure.

Generate-UpperBounds (gUB), the procedure to compute upper bounds, works in the same way except that it computes maxima instead of minima. Formally, **gUB** computes the upper bounds defined as follows:

$$c(s, a) = C(s, a) + \max_{s' \in T(s, a), a' \in A_T(s')} c(s', a'),$$

for every state $s \in [s^a]$ and action a applicable in those states, and $c(s, a) = 0, \forall a \in A_T(s)$ if $s \in G$.

The following theorem establishes the correctness of **gLB** and **gUB**:

Theorem 1. *Let $P = (S, A, T, \gamma, Pr, C, S_0, G)$ be a planning problem that satisfies the assumptions in Section 2. Suppose **gLB** returns a set $\{([s_1^{a_1}], c_1), \dots, ([s_n^{a_n}], c_n)\}$. Then for every i and $s \in [s_i]$, $c_i \leq Q(s, a_i)$.*

*Similarly, if **gUB** returns a set $\{([s_1^{a_1}], c_1), \dots, ([s_n^{a_n}], c_n)\}$, then for every i and $s \in [s_i]$, $c_i \geq Q(s, a_i)$.*

4 Using the Lower and Upper Bounds in MDP Planners

On MDP planning problems that satisfy the assumptions stated in Section 2 the lower and upper bounds computed as above can be used in any MDP planning algorithm that (i) generates the set of applicable actions $A_T(s)$ for every state s it visits, (ii) computes the $Q(s, a)$ values for those actions, (iii) chooses the action that satisfies some optimization criteria (e.g., the action with the minimum $Q(s, a)$ value in a minimization problem), and finally iterates over (i), (ii), and (iii) until it generates a solution to the input MDP planning problem.

Planners like **RTDP** [11] fit directly into this format. **RTDP** repeatedly (i) does a greedy forward search in the state space, and (ii) updates the cost values associated with the visited states in a dynamic-programming fashion. Another example is the well known **Value Iteration (VI)** algorithm [20]. Other examples of planning algorithms that fit into this characterization also include reinforcement learning techniques such as **Q-Learning** [21] and its variants.

Even the optimization techniques such as *Adaptive Multi-Stage Sampling (AMS)* [2] from operations research can be fit into the above format and used as an MDP planning algorithm. **AMS** was designed for solving finite-horizon MDPs with large state spaces [2]. The algorithm can be interpreted as a search of a decision tree, where each node of the tree represents a state (with the root node corresponding to the initial state) and each edge signifies a sampling of a given action. **AMS** employs a depth first search for generating sample paths from the initial state until the given finite horizon is reached, and uses backtracking to estimate the value functions at the visited states. It uses the ideas from multi-armed bandit problems to adaptively sample applicable actions during the search

process. The algorithm's input includes the number N of samples to be used at each stage. Which action to sample at a state s is determined by the current estimated Q function values plus some confidence bounds of all applicable actions in $A_T(s)$. AMS can be used as a planner by modifying its depth-first search so that each search trace of the depth-first search is a sample path that ends when the goals are reached; rather than to stop at a fixed depth as in the original version of the algorithm. For an extensive discussion on AMS, see [2].

During planning, at each state the planner visits, it needs to know the set of all applicable actions in that state. For example, Value Iteration iterates over all of the applicable actions. RTDP chooses whichever action has the currently best Q value. AMS performs adaptive stochastic sampling (where the previous samples affect the future ones) over the set of applicable actions.

We modify an MDP planning algorithm as follows: every time the algorithm needs to know the set of applicable actions $A(s)$ in a state s , we eliminate each action a from $A(s)$ if there is another action a' in $A(s)$ such that the lower bound we computed in the previous section for a is larger than the upper bound computed for a' . In this way, the modified algorithms do not consider any sub-optimal actions in their computation, and in effect, solve a reduced MDP for the original planning problem.

5 Formal Properties and Discussion

Let $P = (S, A, T, \gamma, Pr, C, S_0, G)$ be an MDP planning problem. The modified MDP planning algorithms described above solve a reduced version P_R of P defined by the lower and upper bounds computed as above. The reduced MDP planning problem P_R is the same as P , except that T_R , the state-transition function of P_R , is defined as follows: $T_R(s, a) = \emptyset$ if our action-elimination technique eliminates the action a for the state s , and $T_R(s, a) = T(s, a)$ otherwise. The following theorem establishes that P and P_R have the same optimal solutions:

Theorem 2. *Let $P = (S, A, T, \gamma, Pr, C, S_0, G)$ be an MDP planning problem that satisfies the assumptions in Section 2, and let $P_R = (S, A, T_R, \gamma, Pr, C, S_0, G)$ be the reduced planning problem produced by our action-elimination technique given P as described above. Let π^* and π_R^* be the optimal solutions for P and P_R , respectively. Then $V^{\pi^*}(s_0) = V^{\pi_R^*}(s_0), \forall s_0 \in S_0$.*

Consequently, we can use the bounds computed by the gLB and gUB procedures in order to eliminate actions in the existing MDP algorithms, and still get optimal solutions. Although this is correct for any planning problem that satisfies our assumptions, the gLB and gUB may be time-consuming to compute in some cases. However, there are certain kinds of planning problems in which they can be computed very efficiently and we now discuss two such cases:

- Suppose P can be divided into a set of independent subproblems $\{P_1, \dots, P_k\}$. Then we can compute lower and upper bounds separately for each of the subproblems, generate a solution for each subproblem using those bounds, and combine those solutions to get an solution for P . In this case, the bounds

for each P_i are exactly those that would be computed by calling `gLB` and `gUB` directly on P , hence Theorem 2 guarantees that if we use these bounds to do action elimination, we still get optimal solutions for P by combining the solutions for its subproblems.

- Suppose P can be divided into a sequence of serializable subproblems $\langle P_1, \dots, P_k \rangle$ (see [22]). As before, we can use `gLB` and `gUB` separately for each of the subproblems and generate a solution for each subproblem using those bounds. These policies can then be combined into a near-optimal solution for the original problem.

As an example, consider again the UAV world described in Section 2. The task for the UAV is to start from an initial location and collect photos of all of the objects, by avoiding the threats as much as possible (i.e., by minimizing the harm that may be induced by the threats). Suppose the UAV’s memory is large enough to keep all n photos. Then for each object, collecting its photo is a subproblem of the overall problem, and these subproblems are serializable—i.e., once the UAV collects a photo of one object, it does not lose that photo when it collects the subsequent photos. Thus we can use `gLB` and `gUB` separately, generate a solution for each subproblem, and combine the solutions into a near-optimal solution for the original problem. The reason why the solution is only near-optimal rather than optimal is that the cost of solving a subproblem may depend on the cost of solving another subproblem.

6 Experimental Evaluation

In our experiments, we used the UAV world described above in the paper. As mentioned above, the UAV has five possible actions: `North`, `South`, `East`, `West`, and `CollectPhoto`. The first four actions move the UAV from one location to an adjacent location. The action `CollectPhoto` can be applied only when the UAV is in the same location as an object. We assume that an object cannot be at a threat location in the world.

The task for the UAV is to start from an initial location and collect photos of all of the objects, by minimizing the harm that may be induced by the threats. We assumed that the photos of the objects in a planning problem are collected according to the ordering specified in a planning problem statement.

A threat location is a location such that there is a possibility that the UAV may get destroyed if it enters that location. Each threat location has a level $L = 1, \dots, 5$, from being the least hazardous to being the most hazardous threat to the survival of the UAV. A threat can be at any level at any time; for instance, if a threat is at level $L = 1$ at time t , then it may be in any of 5 levels at time $t + 1$. For each threat level $L = 1, \dots, 5$, the probability Pr_L that the threat is in level L is 0.1, 0.2, 0.4, 0.2, and 0.1, respectively. The threat levels are modeled as the cost of an action that moves the UAV to a threat location and they are taken to be $C_L = 5, 10, 25, 50, \text{ and } 100$, for $L = 1, \dots, 5$.

For our experiments described below, we used the reduced (i.e., abstracted) MDP formulation for the UAV domain mentioned previously and described more

in detail as follows. For each grid location and a possible action a that is applicable in that location, we define an equivalence class of states in the original MDP by a set of state-variable assignments that specify first that the UAV is in that grid location. Also, if there is a threat in the grid location that the UAV will be in after executing the action a , then the state-variable assignments specify the possible threat level of that successor grid location. This partial set of state-variable assignments together specify an equivalence class given the current grid location and the action. The set of all such equivalence classes for each grid location and each possible action in those location constitute our reduced MDP formulation of the original UAV world.

Note that the size of the state space (i.e., the set of all possible states) in the UAV planning problems without any abstraction is in the order of $O(n^2 2^k L^m)$, where n is the number of grids in one dimension of the UAV world, k is the number of objects in the world, m is the number of threat locations, and L is the number of threat levels. This is because a state in the UAV world specifies the location of the UAV, whether the photos of the objects are taken or not, and the combinations of the threat levels. Note that the locations of the objects and the threats are fixed in a UAV problem. For example, if we have a 10×10 grid world with $n = 10$, and we have 5 objects and 5 threat locations with 5 levels for each threat in that world, the size of the state space is $\approx 10^2 \times 2^5 \times 5^5 = 10^7$.

In our reduced formulation based on equivalence classes, the size of the state space is $O(kL|A|n^2)$, where $|A|$ is the number of actions in the planning problem. In particular, in the above setting, the size of the state space is $\approx 5 \times 5 \times 5 \times 10^2 = 12,500$, since (1) for each robot location, there may be five actions applicable, and if there is a threat in the grid that the UAV transitions by applying an action, then that threat can be in five possible levels; and (2) the UAV problems are serializable; i.e., we can process the objects in a particular order, and combine the results. The gLB and gUB procedures exploit this serializability property in the UAV problems, and they further reduce size of the search space of the planners by eliminating those equivalence classes defined by state-action pairs that are guaranteed not to be in a solution.

In our experimental results shown below, these two factors in abstracting away from the state space of the original MDP for the UAV world provided huge performance gains. In the following, we discuss our results in detail. In all of the experiments reported below, we used a HP Pavilion 900MHz laptop with 256MB memory, running Linux Fedora Core 2.

Experiments with AMS. We implemented both the original AMS algorithm and our modified version, called AMS_X, that incorporates the action-elimination technique. We compared the running-time performance of both algorithms in the following two sets of experiments. For both AMS and AMS_X, the number of samples at each node of the search tree was set to $k = 50$.

In the experiments, we fixed the number of threats to 5 and varied the number of objects, $n = 1, \dots, 10$, in a 5×5 grid. For each experimental case $n = 1, \dots, 10$, we randomly generated 50 planning problems by randomly locating the UAV, the objects, and the threats on the grid.

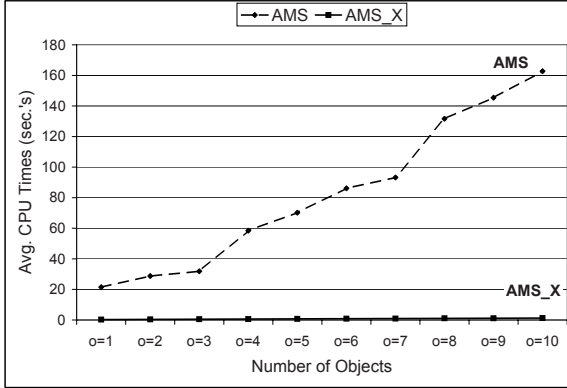


Fig. 3. Running times on UAV path-planning problems with increasing the number of objects, with 5 threats and 5×5 grid. Each data point is the average of 50 problems.

Figure 3 shows the average running times of AMS and AMS_X. The data points in the figure also include the times to compute the lower and the upper bounds on the Q functions of the experimental problems. The running times of both algorithms increase as the number of objects grows, but this increase is much slower in AMS_X due to the action-elimination technique described in the previous section. In particular, in the case of 10 objects, AMS_X was able to solve the problem approximately 130 times faster than AMS.

In order to investigate the effect of the state space size on the running times of our algorithms, we performed another set of experiments, where we fixed both the number of objects and the number of threats to 5, and varied the size of the grid. Table 1 shows the performances of both algorithms, where (–) indicates that a solution was not returned in 30 minutes. It can be seen that AMS_X is far more efficient than AMS. To further test the performance of our approach to larger test instances, we also applied AMS_X to problems with grid sizes of 20×20 and 30×30 . The approximate running times are 7 seconds for the former case, and 20 seconds for the latter case.

Experiments with RTDP. We implemented both the RTDP algorithm described in 1 and our modified version, called RTDP_X, that uses action elimination via bounds. The original RTDP algorithm uses the Bellman Equation (i.e., Equation 2) as described in Section 2, which specifies a deterministic cost value for each state-action pair; i.e., $C(s, a)$ is fixed in advance before the planning process starts. However, the UAV path planning problems require stochastic cost functions defined over state-action pairs. Thus, we used the following variant of the Bellman Equation in both RTDP and our modified version of it:

$$Q(s, a) = [\sum_{L=1, \dots, 5} Pr_L C_L(s, a)] + \gamma \sum_{s' \in T(s, a)} Pr(s, a, s') V(s'), \quad (3)$$

Table 1. Running times of AMS and AMS_X on UAV path-planning problems with 5 objects and 5 threats. n is the size of the grid. Each data point is the average of 50 problems, where (-) indicates that a solution was not returned in 30 minutes.

$n =$	4	5	6	7	8	9	10
AMS	12.3952	70.6416	-	-	-	-	-
AMS_X	0.6492	0.7236	0.9474	0.9442	1.265	1.6542	1.9844

where $L = 1, \dots, 5$ specifies the possible threat levels and Pr_L is the probability that the cost of applying the action a in the state s is the value $C_L(s, a)$, as described above. In effect, we computed the expected cost value for each state-action pair and use those values as deterministic costs in RTDP.

In these experiments, we fixed the number of threats to 3 and varied the number of objects, $n = 1, \dots, 10$, in a 6×6 grid. The reason for the change in the size of the grid and the number of threats is that RTDP, when it commences in its learning of the Q function, sees the treat locations as unpassable obstacles since it uses a greedy search. Thus, we wanted to ensure that in our randomly-generated problems, there are no inaccessible areas on the map such that an object might be inaccessible from where the UAV initially is.

For each experimental case $n = 1, \dots, 10$, we randomly generated 20 planning problems by randomly locating the UAV, the objects and the threats on the grid². For both planners, we used a termination criterion $\epsilon = 10^{-8}$ to guarantee that they do not terminate before finding the best (near-optimal) solution.

Figure 4 shows the average running times of RTDP and RTDP_X. The data points in the figure also include the times to compute the lower and the upper bounds on the Q functions of the experimental problems. The running times of both algorithms increase as the number of objects grows, but this increase is much slower in RTDP_X due to the action-elimination technique described in the previous section. For example, in the experiments with 6 objects, RTDP_X has about 1/2500 of the running time of the original algorithm. Figure 4 does not show any results for RTDP beyond experiments with 6 objects in the planning problems because RTDP in those cases exceeded the time limit of 1 hour CPU time (on 5 of the planning problems with 7 objects, RTDP took around 2.5 hours each, so we did not run the algorithm with the rest of experimental suite). RTDP_X, on the other hand, was able to solve all of the problems in our test suite in the order of few seconds.

Note that although RTDP always chooses the best action among the possible applicable actions in a state, it still does not perform very well because the branching factor in a state is given by all of the possible applicable actions in that state, especially at the early stages of the algorithm when the Q values are converged. Our modified version of RTDP, on the other hand, reduces that branching factor by using the lower and upper bounds during the planning

² The reason that we used less number of random problems in these experiments was to ensure the completion of the experimental suite due to the long run times required the original RTDP algorithm.

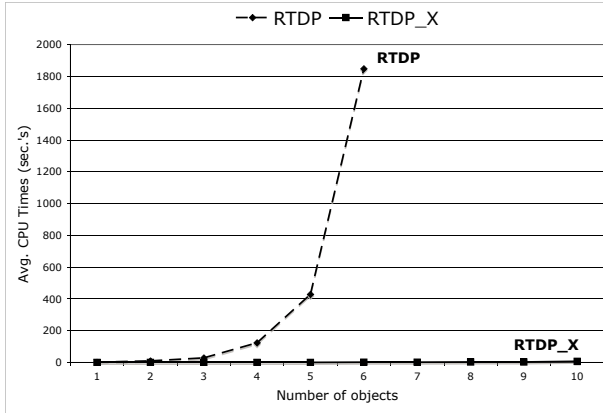


Fig. 4. Running times on UAV path-planning problems with increasing the number of objects, with 3 threats and 6×6 grid. Each data point is the average of 20 problems.

process, since any applicable action in a state whose Q value is not within the bounds is not considered for that state at all.

7 Related Work

Action elimination in MDPs was addressed in early work by MacQueen [13], who used some inequality forms of Bellman’s equation together with bounds on the optimal value function to eliminate non-optimal actions in order to reduce the size of the action space of an MDP. Since then, the method has been applied to several standard MDP solution techniques such as value iteration and policy iteration, see e.g., [16] for a review. Recently, the idea of eliminating non-optimal actions has been explored in [14] in a reinforcement learning context where the explicit mathematical model of the underlying system is unknown. Unlike the action-elimination technique we described in this paper, all these approaches are designed for solving general MDPs, thus they do not exploit the structure of the underlying planning problems. Note that it is possible to combine more than one action-elimination technique, in order to get an even more efficient MDP-based planning framework, and we will investigate such possibilities in the near future.

In addition to the RTDP algorithm described earlier, another similar algorithm that uses a heuristic search for MDP planning is LAO* [11]. LAO* is a generalization of the well-known AO* search algorithm for solving MDPs, using Value Iteration and/or Policy Iteration in order to update the values of the states in the search space. In several experimental studies, RTDP has shown to outperform LAO*, thus, we have not considered LAO* in our experiments.

State aggregation (or state abstraction) has been well studied in artificial intelligence community [23]. Some important advances include the homomorphism method of [24], the bisimulation technique of [25, 7], and the use of Bellman residuals [20] and hierarchical abstractions [26]. A review of these techniques can be

found in [27], where various abstraction techniques are categorized into several general classes. Our notion of equivalent classes in this paper uses ideas from factored MDPs and state aggregation based on similarity between two (groups of) states' cost functions. Thus, our approach can be viewed as a particular technique for model-irrelevance abstraction [27], but it explores and exploits the structure of the underlying state space. Therefore, in cases where the state space admits a factored representation, our technique will in general be more efficient than a generic model-irrelevance abstraction technique. Moreover, it has been shown in [27] that a model irrelevance abstraction will preserve the optimality of the original MDP; this is consistent with the result we obtained in Theorem 2.

Finally, in deterministic search and planning, a very promising approach have been the use of *pattern databases* (*PDBs*) in order to prune the search space. Pattern databases were introduced in [28] as a method for defining effective heuristic functions for search and planning in deterministic domains [29,30]. Our action-elimination technique is similar the notion of computing a PDB in that both our technique and PDB computations are defined by a goal state and an abstraction of the state space. However, an important distinction between our work and the previous work on PDBs is that to the best of our knowledge, PDBs have been designed and used only for deterministic search and planning problems, not for MDP planning problems. However, our algorithms for bound computations are specifically designed with nondeterministic actions and cost functions as in MDPs; e.g., the nondeterministic nature of MDPs was the main reason for using both lower and upper bounds for eliminating non-optimal actions.

8 Conclusions and Future Work

In this paper, we have described a way to improve the efficiency of a large class of existing MDP planning algorithms. Based on the cost function of an MDP planning problem, we first generate a particular partitioning of the MDP represented in a factored form. Then, we compute lower and upper bounds on the Q function of the MDP by doing a backward breadth-first search over the partitioning computed in the previous step. We then take these bounds and incorporate them into any MDP planning algorithm. As a result of this process, the modified planners search a reduced MDP by identifying and eliminating sub-optimal actions, improving the performance.

We have presented theorems showing that our action-elimination technique is correct; i.e., the lower and upper bounds generated by the techniques only eliminate actions that are guaranteed not to be part of an optimal solution, and thus, our technique preserves optimality in the modified MDP planners.

We have presented an experimental evaluation of our ideas in an Unmanned Aerial Vehicle (UAV) path planning domain, using two MDP planning algorithms – namely, RTDP [1] and AMS [2] – that has been demonstrated to be very successful in the automated planning and operations research communities, respectively. Our results demonstrate that our technique can provide significant advantages in speed and scalability. In particular, our modified version of AMS

was able to solve the problem approximately 130 times faster than the original one. Similarly, in the largest problems that RTDP could solve, our modified version of RTDP was approximately 2,500 times faster than the original one.

In the near future, we will conduct an extensive experimental evaluation of our technique, using other MDP planning domains and other MDP planning algorithms such as Q-Learning and LAO*. We are also currently investigating in detail different state-abstraction techniques based on pattern databases [28] and bisimulation [7] that has been very successful in their respective research areas and developing ideas on how to combine them with our technique to generate bounds for action elimination in MDPs. Finally, as we noted in the paper, we are planning to work on combining other existing action-elimination techniques such as the one described in [14] with our framework.

Acknowledgments. We thank to Michael Fu, Steve Marcus, and Dana Nau for several useful discussions on the ideas in this paper and to our anonymous reviewers for their helpful comments. This work was supported in part by NSF grant IIS0412812, DARPA's REAL initiative, and ISR seed funding. The opinions expressed in this paper are those of authors and do not necessarily reflect the opinions of the funders.

References

1. Bonet, B., Geffner, H.: Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming. In: ICAPS-03, pp. 12–21 (2003)
2. Chang, H.S., Fu, M.C., Hu, J., Marcus, S.I.: An adaptive sampling algorithm for solving markov decision processes. *Operations Research* 53(1), 126–139 (2005)
3. Bertsekas, D.P., Castañón, D.A.: Adaptive aggregation methods for infinite horizon dynamic programming. *IEEE Trans. on Automatic Control* 34(6), 589–598 (1989)
4. Dearden, R., Boutilier, C.: Abstraction and approximate decision-theoretic planning. *Artificial Intelligence* 89(1-2), 219–283 (1997)
5. Dean, T., Kaelbling, L.P., Kirman, J., Nicholson, A.: Planning under time constraints in stochastic domains. *Artificial Intelligence* 76(1–2), 35–74 (1995)
6. Boutilier, C., Dearden, R., Goldszmidt, M.: Stochastic dynamic programming with factored representations. *Artificial Intelligence* 121(1-2), 49–107 (2000)
7. Givan, R., Dean, T., Greig, M.: Equivalence notions and model minimization in markov decision processes. *Artificial Intelligence* 147(1-2), 163–233 (2003)
8. Tsitsiklis, J.N., Van Roy, B.: Feature-based methods for large-scale dynamic programming. *Machine Learning* 22, 59–94 (1996)
9. de Farias, D.P., Van Roy, B.: The linear programming approach to approximate dynamic programming. *Operations Research* 51(6), 850–865 (2003)
10. Trick, M., Zin, S.: Spline approximations to value functions: A linear programming approach. *Macroeconomic Dynamics* 1, 255–277 (1997)
11. Hansen, E.A., Zilberstein, S.: LAO*: A Heuristic Search Algorithm that Finds Solutions With Loops. *Artificial Intelligence* 129, 35–62 (2001)
12. Sutton, R., Barto, A.: *Reinforcement Learning: An Introduction*. MIT Press, Cambridge (1998)

13. MacQueen, J.: A modified dynamic programming method for markovian decision problems. *J. Math. Anal. Appl.* 14, 38–43 (1966)
14. Even-Dar, E., Mannor, S., Mansour, Y.: Action elimination and stopping conditions for reinforcement learning. In: *ICML-03* (2003)
15. Boutilier, C., Dean, T., Hanks, S.: Decision theoretic planning: Structural assumptions and computational leverage. *JAIR* 11, 1–94 (1999)
16. Puterman, M.L.: *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley & Sons, Inc, New York (1994)
17. Jum, M., Andrea, R.D.: Path Planning for Unmanned Aerial Vehicles in Uncertain and Adversarial Environments. In: *Cooperative Control: Models, Applications and Algorithms*, Kluwer, Dordrecht (2002)
18. Hanks, S., McDermott, D.: Modeling a dynamic and uncertain world I: Symbolic and probabilistic reasoning about change. Technical Report TR-93-06-10, U. of Washington, Dept. of Computer Science and Engineering (1993)
19. Boutilier, C., Dean, T.L., Hanks, S.: Planning under uncertainty: Structural assumptions and computational leverage. In: Ghallab, Milani (eds.) *New Directions in AI planning*, pp. 157–171. IOS Press, Amsterdam (1996)
20. Bertsekas, D.: *Dynamic Programming and Optimal Control*. Athena Scientific (1995)
21. Watkins, C.J.C.H.: *Learning from Delayed Rewards*. Ph.D. thesis, University of Cambridge (1989)
22. Korf, R.E.: Optimal Path Finding Algorithms. *Search in AI*, pp. 223–267 (1988)
23. Guinchiglia, F., Walsh, T.: A theory of abstraction. *Artificial Intelligence* 57(2-3), 323–390 (1992)
24. Ravindran, B., Barto, A.: Smdp homomorphisms: An algebraic approach to abstraction in semi-markov decision processes. In: *IJCAI-03*, pp. 1011–1016 (2003)
25. Dean, T., Givan, R., Leach, S.: Model reduction techniques for computing approximately optimal solutions for markov decision processes. In: *UAI-97*, pp. 124–131 (1997)
26. Dietterich, T.G.: Hierarchical reinforcement learning with the MAXQ value function decomposition. *JAIR* 13, 227–303 (2000)
27. Li, L., Walsh, T., Littman, M.: Towards a unified theory of state abstraction for mdps. In: *AI and Math-06* (2006)
28. Culberson, J.C., Schaeffer, J.: Efficiently searching the 15-puzzle. Technical report, Department of Computer Science, University of Alberta (1994)
29. Korf, R.E.: Finding optimal solutions to rubikös cube using pattern databases. In: *AAAI-97*, pp. 700–705 (1997)
30. Edelkamp, S.: Planning with pattern databases. In: *Proceedings of the European Conference on Planning (ECP)*, pp. 13–24 (2001)

Model-Based Exploration in Continuous State Spaces

Nicholas K. Jong and Peter Stone

The University of Texas at Austin, Austin TX 78712, USA
{nkj,pstone}@cs.utexas.edu
<http://www.cs.utexas.edu/users/{nkj,pstone}>

Abstract. Modern reinforcement learning algorithms effectively exploit experience data sampled from an unknown controlled dynamical system to compute a good control policy, but to obtain the necessary data they typically rely on naive exploration mechanisms or human domain knowledge. Approaches that first learn a model offer improved exploration in finite problems, but discrete model representations do not extend directly to continuous problems. This paper develops a method for approximating continuous models by fitting data to a finite sample of states, leading to finite representations compatible with existing model-based exploration mechanisms. Experiments with the resulting family of fitted-model reinforcement learning algorithms reveals the critical importance of how the continuous model is generalized from finite data. This paper demonstrates instantiations of fitted-model algorithms that lead to faster learning on benchmark problems than contemporary model-free RL algorithms that only apply generalization in estimating action values. Finally, the paper concludes that in continuous problems, the exploration-exploitation tradeoff is better construed as a balance between exploration and generalization.

1 Introduction

Reinforcement learning (RL) algorithms must balance two motives in selecting actions in controlled systems: exploration and exploitation [1]. Exploratory actions attempt to gather useful data about the system; exploitative actions attempt to maximize expected rewards given the data. Effective exploration remains a challenging research problem, with many RL implementations relying on relatively naive approaches. For example, the seminal Q-learning algorithm [2], which underlies a large fraction of ongoing RL research and most current RL applications, promises asymptotic convergence to an optimal control policy, given an exploration policy that attempts every action in every state infinitely often. In practice, most implementations explore by relying on random deviations from the learned policy. Any sequence of actions is possible in such a scheme, but a sequence becomes exponentially unlikely the longer it deviates from the learned policy. Such inefficient exploration methods help to explain the limitations of applying RL methods to real-world problems, which demand fast convergence to reasonable policies in large or infinite state spaces.

Model-based approaches to RL facilitate more informed exploration by explicitly estimating the dynamics of the system before attempting to estimate the optimal policy. Uncertainty in the learned model can direct the learning algorithm to seek the data most likely to improve exploitation. Such approaches led to the first probabilistic convergence guarantees to near-optimal policies with finite amounts of data [3]. However, this body of research usually relies on tabular representations of models that presuppose discrete problems; model learning for continuous problems has been restricted to the case of deterministic dynamics [4]. Partly for this reason, state-of-the-art algorithms for continuous problems, such as LSPI [5], rely on model-free techniques, which approximate the long-term value of each action in every state directly from data. However, these algorithms still rely on random exploration to acquire this data. To make matters worse, many modern algorithms employ computationally expensive supervised learning mechanisms, which permit them to update their exploration policies very intermittently. In contrast, model-based algorithms typically support incremental updates that permit immediate changes to the exploration policy in response to each new piece of data.

This paper develops model-based algorithms suitable for continuous problems. It addresses the question of how to represent the transition model for an action, which must specify for infinitely many states a successor state distribution over an infinite set. The successor state distribution may be approximated with a finite sample generalized from the data, but this distribution must still be parameterized by the infinite state set. However, if this transition model is used with fitted value iteration [6], an algorithm for computing policies in infinite problems using a finite state sample, then it suffices to represent the transition model explicitly at only a finite number of points. The resulting fitted model permits the application of the simple but effective exploration mechanism from R-MAX [7], a model-based algorithm designed for finite problems.

2 Background

Most RL algorithms assume that the controlled system constitutes a Markov decision process (MDP) [8]. An MDP $\langle S, A, \mathcal{P}, \mathcal{R} \rangle$ comprises a set of states S , a finite set of actions A , a transition function $\mathcal{P} : S \times A \rightarrow \Delta S$, and a reward function $\mathcal{R} : S \times A \rightarrow \mathbb{R}$. For all $s \in S$ and $a \in A$, $\mathcal{P}(s, a) = \mathcal{P}_{sa}$ gives the probability density function over successor states s' given that action a is executed in state s , so $\Pr(s'|s, a) = \mathcal{P}_{sa}(s')$. The reward function gives the expected reward $E[r|s, a] = \mathcal{R}(s, a) = \mathcal{R}_{sa}$ for executing action a in state s . Finally, MDPs satisfy the Markov assumption, which states that the successor state s' and the reward r depend only on s and a . In other words, s' and r are conditionally independent of all other variables given s and a .

In this paper, it will be necessary to reason about the composition of Markovian transition functions, similar to the composition of MDPs in [6]. To this end, suppose $f : Y \rightarrow \Delta X$ and $g : Z \rightarrow \Delta Y$ are transition functions from Y to X and

from Z to Y , respectively. Then the composition of f and g , $f \circ g : Z \rightarrow \Delta X$ is obtained by marginalizing over the values of Y :

$$(f \circ g)_z(x) = \Pr(x|z) = \int_y \Pr(x \wedge y|z) \, dy = \int_y \Pr(x|y, z) \Pr(y|z) \, dy \quad (1)$$

$$= \int_y \Pr(x|y) \Pr(y|z) \, dy \quad (2)$$

$$= \int_y f_y(x) g_z(y) \, dy, \quad (3)$$

where (2) follows from the conditional independence of x from z given y .¹ Note that for the finite case, the composition of transition functions corresponds directly to the multiplication of the appropriate transition matrices.

The optimal value function $V : S \rightarrow \mathbb{R}$ for an MDP specifies the maximum possible expected cumulative reward $V(s)$ given optimal behavior and starting from state $s \in S$. This value function satisfies the Bellman optimality equations: for all $s \in S$,

$$V(s) = \max_{a \in A} \left[\mathcal{R}_{sa} + \int_{s' \in S} \mathcal{P}_{sa}(s') V(s') \, ds' \right]. \quad (4)$$

(A discount factor $\gamma \in [0, 1]$ may be used to ensure that this system has a solution.) Given V , an optimal policy $\pi : S \rightarrow A$ may be defined by $\pi(s) = \operatorname{argmax}_{a \in A} Q(s, a)$, where $Q(s, a) = \mathcal{R}_{sa} + \int_{s' \in S} \mathcal{P}_{sa}(s') V(s') \, ds'$.

3 Model Approximation

Model-based RL algorithms estimate the transition and reward functions \mathcal{P} and \mathcal{R} from experience data. They can then use these estimates relatively directly with (4) to compute the optimal value function and policy. For concreteness, let $s_0, a_0, r_1, s_1, \dots, r_t, s_t$ be the data, with t being the current time step. In episodic tasks, a special state $s^{\text{terminal}} \notin S$ designates the end of an episode. If $s_i = s^{\text{terminal}}$, then a_i and r_{i+1} are undefined, and s_{i+1} is the initial state in the next episode. Additionally, define the set of transition instances $D = \{i \mid 0 \leq i < t \wedge s_i \neq s^{\text{terminal}}\}$. For convenience, we also define subsets of D that condition the data on specific actions and states. Let $D^a = \{i \in D \mid a_i = a\}$ be the set of instances that match action a , let $D_s^a = \{i \in D^a \mid s_i = s\}$ be the set of instances that also matches state s , and let $D_{ss'}^a = \{i \in D_s^a \mid s_{i+1} = s'\}$ be the set of instances that also matches successor state s' .

For finite MDPs, straightforward maximum likelihood estimation of \mathcal{P} and \mathcal{R} is both simple and effective. The model may be computed from D as $\hat{\mathcal{P}}_{sa}(s') = \frac{|D_{ss'}^a|}{|D_s^a|}$ and $\hat{\mathcal{R}}_{sa} = \frac{\sum_{i \in D_s^a} r_{i+1}}{|D_s^a|}$. Initially, D_s^a will be quite small everywhere, but by

¹ $f \circ g$ is not strictly a Markovian transition function, since x is not conditionally independent of y given z , but this subtlety is not relevant to the results of this paper.

the pigeonhole principle the estimate will become quite reliable at some state-action sa . Reliable regions of the model enable a model-based exploration mechanism to direct the agent to regions of the state space where more data is needed, until adequate data exists to estimate the model at every reachable state. This approach to exploration in finite problems is precisely the one taken explicitly in E^3 [3] and implicitly in prioritized sweeping [9] and R-MAX [7].

In very large finite MDPs, the updating the model for only one state-action at a time may require too much data in practice. In infinite MDPs, the algorithm may never visit the same state twice, precluding accurate estimation of the model parameters entirely. One of the primary contributions of this paper is a robust method for approximating the maximum likelihood model from data. The method decomposes the estimated transition function into the composition of components that can be computed easily from the data. A key feature of the final method will be that one of these components generalizes the model across nearby states, but for the sake of clarity the initial description of the decomposition will address the case without generalization.

3.1 Decomposition of the Transition Function

Consider the task of estimating the effect of executing action a in state s , given data D . Instead of directly estimating the transitions as a probability distribution over S as a function of $S \times A$, define a two-stage transition function that first transitions from state-action sa to a state-instance $si \in S \times D$, then from state-instance si to a successor state $s' \in S$. A dynamic Bayesian network of this formulation appears in Fig. 1.

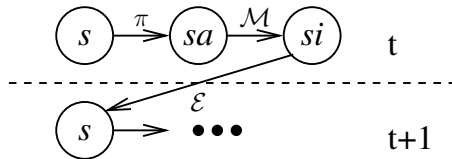


Fig. 1. Dynamic Bayesian network showing the decomposition of the transition function for an approximated MDP. The policy π determines the conditional distribution of sa given s . The model instance transition function \mathcal{M} determines the conditional distribution of si given sa . The action effect transition function \mathcal{E} determines the conditional distribution of s at time $t + 1$ given si from time t .

A model instance transition function $\mathcal{M} : S \times A \rightarrow \Delta(S \times D)$ maps each state-action sa to a state-instance si with probability $\mathcal{M}_{sa}(s, i)$. Intuitively, \mathcal{M} replaces the action component of sa with a specific instance i from the agent’s experience that represents the predicted effect of a . The state-instance si implies that the same thing that happened at time step i will happen again, this time at state s . This transition function thus accounts for the stochasticity in the domain, by replacing the potentially stochastic action a with a specific outcome

i. Note that \mathcal{M} preserves the value of the state s when it transitions a state-action sa to a state-instance si .

In the absence of generalization, the agent has no reason to believe that the action effect at instance i will recur at state s unless $s_i = s$ and $a_i = a$. Hence, the exact model instance transition function is

$$\mathcal{M}_{sa}^{\text{exact}}(s, i) \propto \delta_{ss_i} \delta_{aa_i}, \tag{5}$$

where $\delta_{xy} = 1$ if $x = y$ and 0 otherwise. In the same vein, given state-instance si and $s = s_i$, it must be the case that $s' = s_{i+1}$, since the transition to si accounted for any nondeterminism. The absolute effect transition function $\mathcal{E}_{si}^{\text{abs}} : S \times D \rightarrow \Delta S$ reflects this expectation:

$$\mathcal{E}_{si}^{\text{abs}}(s') = \delta_{s' s_{i+1}}. \tag{6}$$

It can be verified that the composition of $\mathcal{E}_{si}^{\text{abs}}$ and $\mathcal{M}_{sa}^{\text{exact}}$ yields the maximum likelihood estimator for \mathcal{P} given above:

$$\begin{aligned} (\mathcal{E}^{\text{abs}} \circ \mathcal{M}^{\text{exact}})_{sa}(s') &= \sum_{i \in D} \delta_{s' s_{i+1}} \frac{\delta_{ss_i} \delta_{aa_i}}{\sum_{i \in D} \delta_{ss_i} \delta_{aa_i}} \\ &= \sum_{i \in D} \frac{\delta_{s' s_{i+1}} \delta_{ss_i} \delta_{aa_i}}{|D_s^a|} \\ &= \frac{|D_{ss'}^a|}{|D_s^a|} \\ &= \hat{\mathcal{P}}_s^a(s'). \end{aligned}$$

3.2 Model Generalization

The preceding section presented a novel computation of the exact maximum likelihood estimator for \mathcal{P} , but as discussed at the beginning of Sect. 3 in very large or infinite MDPs this estimator is impractical. This section describes alternative definitions of the model instance transition function \mathcal{M} and action effect transition function \mathcal{E} that are more useful in continuous problems. To predict the effects of actions at infinitely many states given only finite data, the learned model must use some form of generalization and hence inductive bias. This section of the paper places additional assumptions on the state space of the MDP to be learned. In particular, it assumes the state space is a bounded subset of some Euclidean space, and it assumes that nearby states tend to induce similar dynamics and reward for each action.

The approximate model lifts the restriction that the exact model imposes, allowing a state-action sa to transition to a state-instance si such that $s_i \neq s$. However, the model weights each transition according to a decreasing function of the euclidean distance $|s_i - s|$ between s_i and s . This paper uses Gaussian weighting:

$$\mathcal{M}_{sa}^{\text{approx}}(s, i) \propto \delta_{aa_i} e^{-\left(\frac{|s-s_i|}{b}\right)^2}, \tag{7}$$

where b is a parameter that controls the breadth of generalization across the state space. This parameter critically affects learning performance. A large degree of generalization permits very rapid learning, but in some cases overgeneralization can prevent the algorithm from ever finding a good policy.

Even moderate amounts of generalization suggest a modification to the absolute action effect transition function \mathcal{E}^{abs} defined in Sect. 3.1, as shown in Fig. 2. For a given state-instance s_i , predicted the successor state to be s_{i+1} , the actual successor state for the instance i , makes less sense the farther s_i is from s . Early experiments demonstrated that the breadth of generalization can be quite large relative to the distance traveled in one time step. Otherwise, the amount of data required may be prohibitive, despite potential regularities in the system’s dynamics. For example, a mobile robot whose state includes its pose should be able to generalize its action model over large regions of free space. The relative action effect transition function thus attempts to isolate for a given instance i the contribution of the action a_i from the contribution of the state s_i on the successor s_{i+1} :

$$\mathcal{E}_{s_i}^{\text{rel}}(s') = \begin{cases} 1, & \text{if } s_{i+1} = s^{\text{terminal}} \wedge s' = s^{\text{terminal}} \\ 1, & \text{if } s' = s + (s_{i+1} - s_i) \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

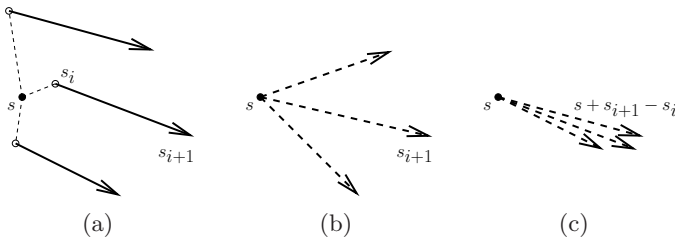


Fig. 2. The action effect transition function \mathcal{E} can determine the fidelity of the approximated model to the true dynamics. (a) Approximating the effect of some action a at a given state s using three nearby instances. (b) Absolute action effects predict transitions to the exact successor states previously visited. (c) Relative action effects better capture the dynamics of the system by applying the appropriate vectors to the present state s .

Finally, the approximation of the reward function $\mathcal{R}^{\text{approx}}$ is similar to the approximation of the model approximation transition function. For a given state-action sa , the approximated expected reward is a weighted average of the rewards for the instances used to approximate a near s :

$$\mathcal{R}_{sa}^{\text{approx}} = \sum_{i \in D} r_{i+1} \mathcal{M}_{sa}^{\text{approx}}(s, i). \tag{9}$$

4 Fitted-Model Learning Algorithms

Section 3 gave an approximation of the transition and reward functions for an unknown continuous-state MDP, but a complete model-based algorithm also requires a practical method for computing the value function from the model and an exploration mechanism. This section integrates the contributions of Sect. 3 with existing algorithms that play each of these roles.

4.1 Fitted Models

The approximate model instance transition function $\mathcal{M}^{\text{approx}}$ induces a continuous MDP $\langle S, A, \mathcal{P}^{\text{approx}}, \mathcal{R}^{\text{approx}} \rangle$, with $\mathcal{P}^{\text{approx}} = \mathcal{E}^{\text{rel}} \circ \mathcal{M}^{\text{approx}}$ and $\mathcal{R}^{\text{approx}}$ given by (9). Computing the optimal value function for even this approximate model is impossible in general, since the transition and reward functions still vary continuously over the infinite state-action space $S \times A$.

Fitted value iteration [6] provides an algorithm for approximating the optimal value function of continuous-state MDPs. The algorithm uses a finite sample $X \subset S$ of states to represent the value function, for an arbitrary value function approximation scheme that represents the value of any state $s \in S$ as some weighted average of the values of X . That is, the function approximator must compute the value of a state $s \in S$ as $V(s) = \sum_{x \in X} \mathcal{F}_s(x)V(x)$, where $\sum_{x \in X} \mathcal{F}_s(x) = 1$ and $\mathcal{F}_s(x) \geq 0$. In other words, $\mathcal{F} : S \rightarrow \Delta X$ must be a transition function that transitions every state in S to one of the states in the finite sample X . Then interleaving steps of value iteration with fitting the value function to the function approximation scheme is equivalent to applying standard value iteration [10] to the derived MDP $\langle X, A, \mathcal{F} \circ \mathcal{P}, \mathcal{R} \rangle$, as diagrammed in Fig. 3 and 4. This equivalence ensures that function approximation does not cause the value function computation to diverge.

To compute a value function for the approximate model defined in (7) and (9), it suffices to substitute $\mathcal{P}^{\text{approx}}$ for \mathcal{P} and $\mathcal{R}^{\text{approx}}$ for \mathcal{R} in fitted value iteration. Approximating the value function for the learned model is thus equivalent to

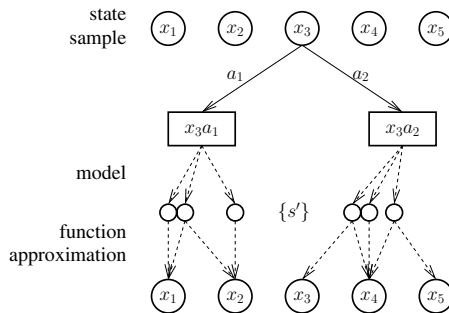


Fig. 3. This diagram shows a continuous MDP with two actions being fitted to a state sample of size 5. For clarity, only the transitions for state s_3 are shown.

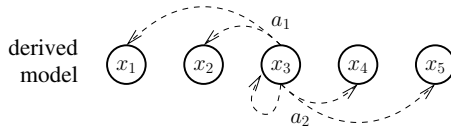


Fig. 4. This diagram shows the finite MDP derived from fitting the continuous MDP in Fig. 3.

computing the exact value function for the finite MDP $\langle X, A, \mathcal{F} \circ \mathcal{E} \circ \mathcal{M}, \mathcal{R} \rangle$. Fig. 5 illustrates this decomposition.

Many function approximation schemes are possible for choosing X and defining \mathcal{F} . In all the experiments described in this paper, X is a uniform grid spanning the state space, and $\mathcal{F}_{s'}(x)$ gives the coefficients for multilinear interpolation of s' from the 2^d corners of the hypercube containing s' , where d is the dimensionality of the state space. Preliminary experiments showed that this simple function approximation scheme performed better than a number of alternatives, including instance-based approaches that added either visited states s_t to X or predicted successors s' to X as necessary.

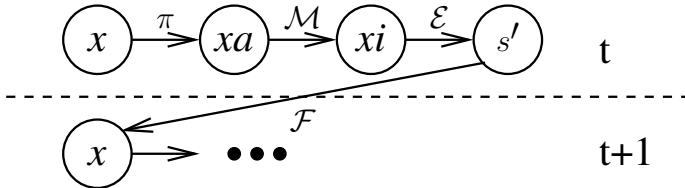


Fig. 5. Dynamic Bayesian network showing the decomposition of transitions in the derived MDP, solved using standard value iteration, into components of the model approximation.

4.2 Fitted R-Max

Section 4.1 showed how to estimate the optimal value function from data by first approximating a model, but one of the primary motivations behind extending model-based methods to continuous problems is to take advantage of intelligent exploration methods. This section describes one simple but effective model-based algorithm for finite problems and shows how to incorporate its exploration mechanism into fitted value iteration.

R-MAX is a relatively simple model-based algorithm that implements a standard principle of exploration: optimism in the face of uncertainty [7]. It maintains maximum-likelihood estimates of the model parameters $\hat{\mathcal{P}}_{sa}(\cdot)$ and $\hat{\mathcal{R}}_{sa}$ for every state s and action a , but it only employs these estimates given sufficient data to have confidence in their accuracy. Let $n(s, a)$ denote the number of times action a has been executed in state s . Then R-MAX estimates the value function using

$$\hat{Q}(s, a) = \begin{cases} V^{\max} & \text{if } n(s, a) < m \\ \hat{\mathcal{R}}_{sa} + \sum_{s' \in S} \hat{\mathcal{P}}_{sa}(s') \hat{V}(s') & \text{if } n(s, a) \geq m \end{cases} \quad (10)$$

where $\hat{V} = \max_{a \in A} \hat{Q}(s, a)$, V^{\max} is an upper bound on the value function,² and m is a constant. The modified Bellman equations can still be solved using a standard MDP planning algorithm, such as value iteration.

The optimistic value function explicitly rewards the algorithm for executing actions in uncertain states. The parameter m determines the amount of exploration required before the algorithm is certain about the effects of a state-action pair. Furthermore, augmenting the value function in this manner causes the agent to seek out states that are either actually high in value or where “exploration bonuses” are available for executing unfamiliar state-actions. [11] showed how the exploration threshold m relates to the likely error in the estimates $\hat{\mathcal{P}}$, leading to bounds on the amount of exploration required before converging to a probably approximately optimal policy.

Although the use of generalization in approximating a fitted model eliminates such guarantees of convergence to optimal behavior, the exploration mechanism of R-MAX can still be applied to fitted value iteration simply by substituting in the parameters of the derived finite MDP into (10) and appropriately defining an approximation of $n(s, a)$. This latter quantity denotes the number of times a was executed in state s , so the logical analog is the sum of the unnormalized kernel values used to weight the transitions from a state s to each $i \in D$:

$$\tilde{n}(s, a) = \sum_{i \in D} \delta_{aa_i} e^{-\left(\frac{|s-s_i|}{b}\right)^2}. \quad (11)$$

Thus $n(s, a)$ now counts both the actual data for a at s as well as “partial” data generalized from executions of a near s . The fitted R-MAX algorithm thus computes the following value function:

$$Q(s, a) = \begin{cases} V^{\max} & \text{if } n(s, a) < m \\ \mathcal{R}_{sa}^{\text{approx}} + \sum_{x' \in X} \left(\mathcal{F} \circ \mathcal{E}^{\text{relative}} \circ \mathcal{M}^{\text{approx}} \right)_{sa}(x') V(x') & \text{if } n(s, a) \geq m \end{cases} \quad (12)$$

At each time step, fitted R-MAX adds the just observed transition to D , updates $\mathcal{M}^{\text{approx}}$ and \mathcal{F} , and then applies value iteration to solve (10). The algorithm then behaves greedily with respect to $Q(s_t, \cdot)$. In practice, the composition $\tilde{\mathcal{P}} = \mathcal{F} \circ \mathcal{E}^{\text{relative}} \circ \mathcal{M}^{\text{approx}}$ need not be recomputed after each time step. By caching the appropriate intermediary values, the finite transition function $\tilde{\mathcal{P}} : X \times A \rightarrow \Delta X$ can be repaired to reflect each new instance i . In the same spirit, prioritized sweeping [9] can be used to update the value function efficiently to reflect changes in the approximate model.

² [7] reasons with R_{\max} , an upper bound on the one-step reward, since its version of the algorithm computes finite-horizon value functions.

5 Experimental Results

Fitted R-MAX learns with good data efficiency by using a combination of model-based exploration and stable function approximation. This section describes experiments demonstrating that fitted R-MAX converges more rapidly to near-optimal policies than several other recent RL algorithms evaluated on some benchmark problems with continuous state spaces. It then examines the importance of the relative action effect transition function \mathcal{E}^{rel} compared to the absolute version \mathcal{E}^{abs} . Finally, it investigates the importance of the generalization breadth parameter, b .

5.1 Implementation Details

A primary practical concern for any instance-based algorithm is computational complexity. The computationally intensive step of fitted R-MAX is the incremental update to the derived finite model. In general, these steps require running time linear in the size of D , which is equal to the number of times the agent has acted.

The experimental implementation achieves a substantial reduction in the constant factor of this $O(|D|)$ running time by observing that the each newly sampled transition only changes the model appreciably in a local region of the state space. It sets the minimum nonzero value of the (unnormalized) Gaussian weighting to 0.01 in (7). Thus the addition of a new transition from s only affects those sample states $x \in X$ within distance $b\sqrt{-\log 0.01} = 2.146b$ from s . The implementation also prunes each averager ϕ so that the smallest nonzero value of $\mathcal{M}_{sa}^{\text{approx}}(s, i)$ is 0.01 (and renormalizes the remaining values), bounding to 100 the number of instances used to approximate s . Note that this pruning does not bias the approximation, which essentially becomes k -nearest neighbors with $k = 100$ and Gaussian weighting whenever sufficient data exists to override optimism. The precise thresholds used to prune did not significantly affect the performance of the algorithm.

5.2 Benchmark Performance

This section compares the performance of fitted R-MAX to algorithms submitted to the RL benchmarking workshop held at NIPS 2005 [12]. This event invited researchers to implement algorithms in a common interface for online RL. Participants computed their results locally, but direct comparisons are possible due to the standardized environment code, which presents the same sequence of initial states to each algorithm. This sections examines two of the benchmark domains and gives the fitted R-MAX parameters used to solve them. It then evaluates the performance of fitted R-MAX against selected algorithms.

Mountain Car. In the Mountain Car simulation [1], an underpowered car must escape a valley (Fig. 6a) by backing up the left slope to build sufficient energy

parameterized by other researchers, were among the most competitive submitted. One is a model-based approach applied to a fixed discretization of the state space. This algorithm employed the same exploration mechanism as Prioritized Sweeping, but it lacked the instance-based representation and averager-based generalization of fitted R-MAX. Least Squares Policy Iteration [5] is similar to fitted R-MAX in that it uses a given sample of transitions to compute the parameters of a function approximator that best approximates the true value function. However, LSPI relies on random exploration and a fixed set of kernels to represent the state space. XAI (eXplore and Allocate, Incrementally) is a method that represents the value function with a network of radial basis functions, allocated online as the agent reaches unexplored regions of the state space [12]. It thus resembles fitted R-MAX in its instance-based use of Gaussian weighting for approximation, but XAI is a model-free method that uses gradient descent and Sarsa(λ) to update the value function. None of these algorithms achieves the same level of performance as fitted R-MAX, which combines instance-based model approximation, stable function approximation, and model-based exploration.

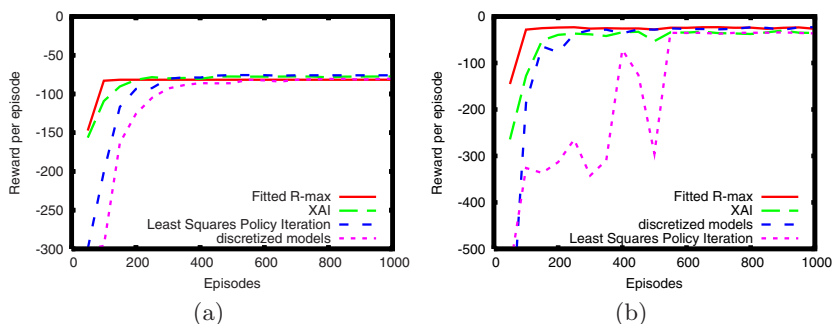


Fig. 7. Learning curves for (a) Mountain Car and (b) Puddle World

5.3 Ablation Study

This section illustrates the benefit of fitted R-MAX’s approach to model-based RL in infinite systems. It compares three algorithms. The first is fitted R-MAX, employing the relative action effect transition function \mathcal{E}^{rel} given in (8). The second is a version of fitted R-MAX that uses the absolute action effect transition function \mathcal{E}^{abs} given in (6), to measure the importance of action effect component of the transition function. The third algorithm is the original discrete R-MAX algorithm [7], to measure the importance of the novel decomposition of the transition function.

Figure 8 shows the performance of each algorithm, averaged over 50 independent trials in the Mountain Car domain. This implementation of Prioritized Sweeping uses the same parameters as the finite model-based algorithm submitted to the NIPS workshop: it discretizes each state dimension into 100 intervals and uses $m = 1$. Fitted R-MAX used the same parameters described in Sect. 5.2

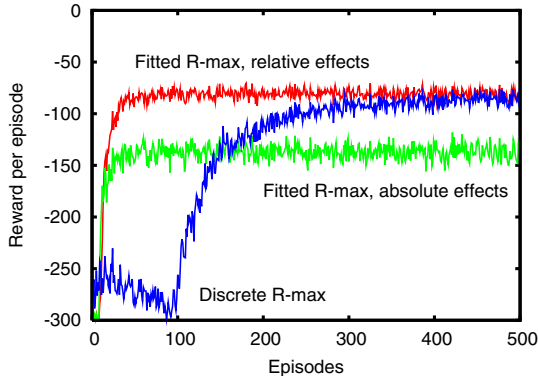


Fig. 8. Learning curves for Mountain Car. Each curve is the average of 50 independent trials.

Absolute-transition fitted R-MAX converges much more quickly than discrete Prioritized Sweeping, but at the expense of converging to suboptimal policies. Further experimentation has shown that decreasing b improves the average quality of the final policy but quickly decreases the learning speed of the algorithm. The standard version of fitted R-MAX uses the more accurate relative transition generalization to preserve fast convergence while achieving near-optimal policies in this domain. For comparison, Figure 9 illustrates typical learned policies for both versions of fitted R-MAX. An optimal policy would execute **forward** roughly when the velocity is positive, in the upper half of the state-space diagram, and it would execute **reverse** roughly when the velocity is negative, in the lower half of the state-space diagram. This run of absolute-transition fitted R-MAX incorrectly selects **reverse** in a large region with positive velocity. Inspection of the relevant states revealed that the local neighborhood of the sample S^{reverse} happened to contain more high-value states. The absolute transition model incorrectly concluded that the **reverse** action would transition to this higher-value region; the relative transition model correctly concluded that this action decreases the value of any state in the neighborhood.

6 Discussion and Related Work

The primary contribution of this paper is its integration of model-based exploration with stable function approximation. Fitted R-MAX extends the data efficiency of model-based methods to continuous systems, which previously presented the difficulty of representing continuous models. [4] addressed this problem in the deterministic case, also using locally weighted learning from instances. Their application of locally weighted regression estimated the average successor state for each state-action pair; fitted R-MAX approximates the distribution over successor states and thus copes with forms of stochasticity beyond simple noise.

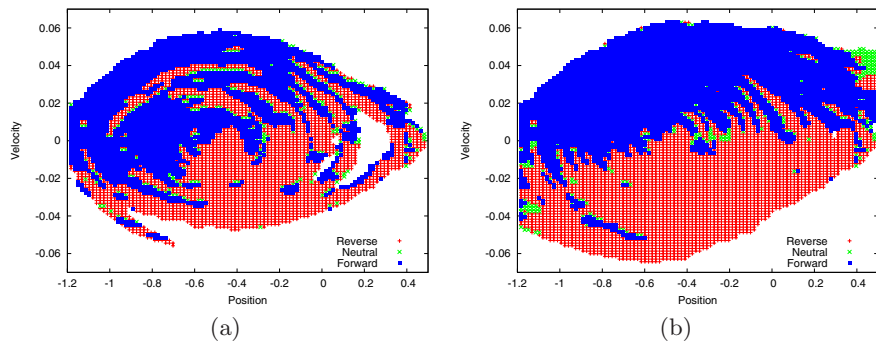


Fig. 9. Mountain-Car policies learned using (a) absolute-transition fitted R-MAX and (b) standard fitted R-MAX. The solid region of the state space indicates where the policy selects the **forward** action; the hatched region indicates where it selects the **reverse** action.

They also did not address the issue of exploration in continuous systems. Fitted R-MAX permits the application of intelligent exploration mechanisms originally designed for finite systems. It employs the same mechanism as Prioritized Sweeping [9] and R-MAX [7], perhaps opening the door for generalizing the latter algorithm’s polynomial-time PAC convergence guarantees to certain continuous systems.

Introducing model-based reasoning to function approximation also provides novel insight into the problem of generalizing from finite data to knowledge of an infinite system. Most approaches to function approximation rely on a static scheme for generalizing the value function directly, despite the difficulty in intuiting the structure of value functions. Fitted R-MAX explicitly generalizes first in a model of the system, where intuitions may be easier to represent. For example, a high degree of generalization is possible in the model for Mountain Car, since the effect of an action changes smoothly with the current state. In contrast, the optimal value function for this system includes large discontinuities in locations that are impossible to predict without first knowing the optimal policy: the discontinuity separates those regions of the state space where the agent has sufficient energy to escape the valley and from those regions where it must first build energy. Approaches that only generalize the value function must use little enough generalization to represent this discontinuity accurately; fitted R-MAX uses a learned model to generalize both broadly and accurately.

7 Conclusion

Reinforcement learning in infinite systems requires accurate generalization from finite data, but standard approaches only apply generalization directly to the value function. Many systems of interest exhibit more intuitive structure in their one-step dynamics than in the optimal value function. This observation suggests

a model-based solution that generalizes first from data to a model. Fitted-model algorithms such as fitted R-MAX apply generalization both to the model and to the value function. They derive a finite representation of the system that both allows efficient planning and intelligent exploration. These attributes allow fitted R-MAX to learn some standard benchmark systems more efficiently than many contemporary RL algorithms.

References

1. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
2. Watkins, C.: Learning From Delayed Rewards. PhD thesis, University of Cambridge (1989)
3. Kearns, M., Singh, S.: Near-optimal reinforcement learning in polynomial time. In: Proceedings of the Fifteenth International Conference on Machine Learning, pp. 260–268 (1998)
4. Atkeson, C.G., Moore, A.W., Schaal, S.: Locally weighted learning for control. *Artificial Intelligence Review* 11, 75–113 (1997)
5. Lagoudakis, M.G., Parr, R.: Least-squares policy iteration. *Journal of Machine Learning Research* 4, 1107–1149 (2003)
6. Gordon, G.J.: Stable function approximation in dynamic programming. In: Proceedings of the Twelfth International Conference on Machine Learning (1995)
7. Brafman, R.I., Tenenbholz, M.: R-max – a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research* 3, 213–231 (2002)
8. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., West Sussex, England (1994)
9. Moore, A.W., Atkeson, C.G.: Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning* 13, 103–130 (1993)
10. Littman, M.L., Dean, T.L., Kaelbling, L.P.: On the complexity of solving Markov decision problems. In: Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence (1995)
11. Kekade, S.M.: On the Sample Complexity of Reinforcement Learning. PhD thesis, University College London (2003)
12. Dutech, A., Edmunds, T., Kok, J., Lagoudakis, M., Littman, M., Riedmiller, M., Russell, B., Scherrer, B., Sutton, R., Timmer, S., Vlassis, N., White, A., White-son, S.: Reinforcement learning benchmarks and bake-offs II (2005) <http://www.cs.rutgers.edu/~mlittman/topics/nips05-mdp/bakeoffs05.pdf>
13. Sutton, R.S.: Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems* 8 (1996)

Active Learning of Dynamic Bayesian Networks in Markov Decision Processes

Anders Jonsson¹ and Andrew Barto²

¹ Department of Information and Communication Technologies

Universitat Pompeu Fabra
Passeig de Circumval·lació, 8
08003 Barcelona, Spain

anders.jonsson@upf.edu

² Autonomous Learning Laboratory

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

Abstract. Several recent techniques for solving Markov decision processes use dynamic Bayesian networks to compactly represent tasks. The dynamic Bayesian network representation may not be given, in which case it is necessary to learn it if one wants to apply these techniques. We develop an algorithm for learning dynamic Bayesian network representations of Markov decision processes using data collected through exploration in the environment. To accelerate data collection we develop a novel scheme for active learning of the networks. We assume that it is not possible to sample the process in arbitrary states, only along trajectories, which prevents us from applying existing active learning techniques. Our active learning scheme selects actions that maximize the total entropy of distributions used to evaluate potential refinements of the networks.

1 Introduction

Existing solution techniques for Markov decision processes, or MDPs, scale poorly to tasks with large state spaces. A major research challenge is to develop techniques that exploit the structure of a task and reduce the size of the state space. A common type of structure is factored state, which means that the available information belongs to distinct categories. For example, a robot navigating through a building can usually distinguish between its location, the object it is holding, and the energy level of its battery, instead of perceiving the current situation as a single observation. Factored MDPs use a set of state variables to represent the state in a way that is more appropriate for tasks of this type. Dynamic Bayesian networks, or DBNs [1], are particularly well suited for exploiting structure in factored MDPs by capturing conditional independence between state variables as a result of executing actions. Several researchers have developed algorithms for solving factored MDPs that exploit structure expressed by DBNs [2,3,4,5,6].

It is unrealistic to assume that a DBN model is always available prior to solving an MDP. We address the non-trivial problem of learning DBNs from experience. There

exist algorithms in the literature for learning the structure of Bayesian networks [7,8,9]. However, these algorithms assume that a data set is given, whereas solution techniques for MDPs typically have to gather data in the form of transitions and reward through interaction with the environment. The complexity of learning DBNs depends heavily on the time it takes to collect data. It is possible to accelerate data collection by selecting high-quality data instances through a process called active learning. There exist several techniques for active learning of Bayesian networks [10,11,12]. These techniques perform experiments by clamping a subset of the variables to fixed values and sampling over the remaining variables.

A robot exploring its environment for the first time cannot transport itself to any location instantaneously. Instead, it must wander around to try the effect of different actions in different places. In this work, we assume that it is only possible to sample MDPs along trajectories, not in arbitrary states. In other words, the only way to gather information about transitions and reward is by repeatedly executing an action in the current state. Since it is not possible to simulate the effect of actions in hypothetical states, we cannot perform experiments by clamping a subset of the variables to fixed values. Consequently, we cannot apply existing techniques for active learning. However, there is still an opportunity to perform active learning of DBNs in factored MDPs. Because the DBN model of a factored MDP consists of one DBN for each action, by selecting an action we effectively select a DBN to collect data for. As a consequence, we can consider policies for action selection whose aim is to gather data as quickly and efficiently as possible. As far as we know, there exists no previous work for learning DBN models of factored MDPs under these assumptions.

1.1 Overview of our Work

We use trees to represent the conditional probabilities of the DBNs, and develop an algorithm that implicitly learns the DBNs by growing the conditional probability trees. Our algorithm collects data instances by executing actions and grows the trees as soon as a minimum number of data instances correspond to each relevant value of each split variable. The minimum number is defined by a threshold parameter, and potential refinements are evaluated as soon as the threshold is exceeded. The algorithm uses the Bayesian Information Criterion (BIC) [13] and the likelihood-equivalent Bayesian Dirichlet metric (BDe) [9] to evaluate potential refinements. We assume that no data is available to begin with and develop a technique for active learning of DBNs to accelerate data collection. The time to collect data is minimized if the distribution of data instances across values of each potential split variable is perfectly uniform. We use the entropy of the distributions to measure uniformity and select actions that maximize the total entropy of the distributions.

In some tasks, the BIC and BDe scores fail to detect most of the refinements necessary to learn an accurate DBN model. This typically happens when the effect of actions depends on many state variables. Since the BIC and BDe scores penalize trees with many leaves, the algorithm prefers to keep the size of the trees small instead of continuing to refine the trees. This is a serious issue since algorithms that take advantage of DBNs to solve factored MDPs depend on an accurate DBN model. We address this issue by applying regularization [14] to the BIC score. The BIC score is composed of a

log likelihood term and a penalty term. This quantity fits nicely into the regularization framework if we multiply the penalty term by a parameter λ . Results show that varying λ can increase the accuracy of the learned DBN model.

Our work is related to the problem of exploration in reinforcement learning [15]. Existing exploration techniques do not learn DBN models of MDPs. Since there exist several efficient algorithms that use DBNs to solve factored MDPs, there is a benefit to learning this representation. Ours is an undirected approach that does not require enumeration of the state space, as opposed to directed exploration, which maintains relevant information for each state. Since we want to scale to large state spaces, we do not want to store quantities whose size is proportional to the number of states.

2 Bayesian Networks

Let \mathbf{X} be a set of discrete variables, and let \mathbf{x} be an assignment of values to the variables in \mathbf{X} . Let $f_{\mathbf{Y}}$, $\mathbf{Y} \subseteq \mathbf{X}$, be a projection such that if \mathbf{x} is an assignment to \mathbf{X} , $f_{\mathbf{Y}}(\mathbf{x})$ is \mathbf{x} 's assignment to \mathbf{Y} . A Bayesian network (BN) $B = \langle G, \theta \rangle$ consists of a directed acyclic graph G with one node per variable $X_i \in \mathbf{X}$ and a set of parameters θ defining the conditional probabilities of the variables. The joint probability distribution of the variables is given by:

$$P(\mathbf{x}) = \prod_i P(X_i = f_{\{X_i\}}(\mathbf{x}) \mid \mathbf{Pa}(X_i) = f_{\mathbf{Pa}(X_i)}(\mathbf{x})),$$

where $\mathbf{Pa}(X_i) \subset \mathbf{X}$ is the subset of parent variables of X_i , i.e., variables with edges to X_i in G , and the probabilities $P(X_i = f_{\{X_i\}}(\mathbf{x}) \mid \mathbf{Pa}(X_i) = f_{\mathbf{Pa}(X_i)}(\mathbf{x}))$ are defined by parameters in θ .

A dynamic Bayesian network, or DBN [11], is a Bayesian network that models the evolution of a set of variables in a temporal process. The directed acyclic graph of a DBN has two layers of nodes: one layer representing the current values of the variables, and one layer representing the next values of the variables. The edges between layers are unidirectional and always point from the current layer to the next layer. There can also be edges between nodes within a layer.

Structure learning is the problem of finding the BN that best fits a data set $D = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$. A common approach is to compute the posterior probability distribution $P(B \mid D)$ over BNs and choose the BN that maximizes $P(B \mid D)$. Two common approximations of $P(B \mid D)$ are the Bayesian Information Criterion (BIC) [13] and the likelihood-equivalent Bayesian Dirichlet metric (BDe) [9]. From Bayes theorem it follows that $P(B \mid D) \propto P(D \mid B)P(B)$. The BIC score makes the approximation

$$\log[P(D \mid B)P(B)] \approx L(D \mid B) - \frac{|\theta|}{2} \log |D|, \quad (1)$$

where $L(D \mid B)$ is the log likelihood of D given B . If the data set D contains no missing values, the log likelihood decomposes as

$$L(D \mid B) = \sum_i \sum_j \sum_k N_{ijk} \log \theta_{ijk},$$

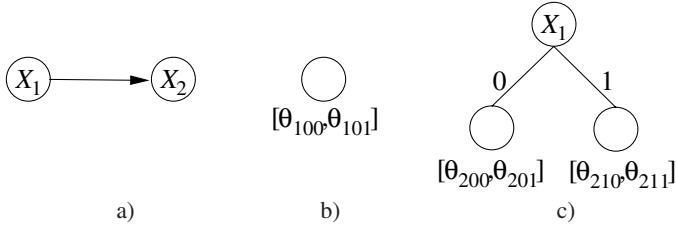


Fig. 1. a) Graph G of a BN with two variables; conditional probability trees for b) X_1 , c) X_2

where N_{ijk} is the number of data points $\mathbf{x} \in D$ such that $f_{\mathbf{Pa}(X_i)}(\mathbf{x}) = j$ and $f_{\{X_i\}}(\mathbf{x}) = k$, and $\theta_{ijk} = P(X_i = k \mid \mathbf{Pa}(X_i) = j)$. The log likelihood is maximized for $\theta_{ijk} = N_{ijk} / \sum_k N_{ijk}$. The BDe score makes the approximation

$$P(D \mid B)P(B) \approx \prod_i \prod_j \frac{\Gamma(\sum_k N'_{ijk})}{\Gamma(\sum_k [N'_{ijk} + N_{ijk}])} \prod_k \frac{\Gamma(N'_{ijk} + N_{ijk})}{\Gamma(N'_{ijk})}, \quad (2)$$

where N'_{ijk} are hyperparameters of a Dirichlet prior and $\Gamma(x)$ is the Gamma function.

Finding the BN with highest BIC or BDe score is NP-complete [16]. However, both scores decompose into a sum of terms for each variable X_i and each value j and k (we need to take the logarithm of BDe first). The score only changes locally when we add or remove edges between variables in G . Researchers have developed hill-climbing algorithms that perform greedy search to find high-scoring BNs by repeatedly adding or removing edges between variables in G [7,9]. These algorithms have been extended to DBNs [8].

As an example, consider a BN with two binary variables X_1 and X_2 . Assume that we have collected three data points (0, 0), (0, 1), and (1, 1). Also assume that the BN has an edge from X_1 to X_2 , and that we use trees to store the conditional probabilities of X_1 and X_2 . Figure 1 shows the graph G of the BN as well as the conditional probability trees for X_1 and X_2 . Since X_1 has no parents in G , the count N_{100} simply indicates the number of data points that assign 0 to X_1 . In this case, $N_{100} = 2$ and $N_{101} = 1$. The log likelihood is maximized for $\theta_{100} = N_{100} / (N_{100} + N_{101}) = 2/3$ and $\theta_{101} = 1/3$. One data point, (0, 0), assigns the value 0 to X_1 and 0 to X_2 , so $N_{200} = 1$. Likewise, $N_{201} = 1$, $N_{210} = 0$, and $N_{211} = 1$. The log likelihood is maximized for $\theta_{200} = N_{200} / (N_{200} + N_{201}) = 1/2$, $\theta_{201} = 1/2$, $\theta_{210} = 0$, and $\theta_{211} = 1$.

The BIC score for the BN is given by the expression

$$\begin{aligned} & \sum_i \sum_j \sum_k N_{ijk} \log \theta_{ijk} - \frac{|\theta|}{2} \log |D| = \\ & = 2 \log \frac{2}{3} + 1 \log \frac{1}{3} + 1 \log \frac{1}{2} + 1 \log \frac{1}{2} + 0 + 1 \log 1 - \frac{6}{2} \log 3. \end{aligned}$$

Note that each leaf of the conditional probability tree for variable X_i contributes to the BIC score with a term $\sum_k N_{ijk} \log \theta_{ijk} - \frac{|Dom(X_i)|}{2} \log |D|$, where $Dom(X_i)$ is the domain of X_i and j is the assignment of values to the parents of X_i in G as indicated

by the path from the root to the leaf. Also note that the contribution from each leaf is smaller than 0, and that it is maximized when all data points assign the same value to X_i , in which case the first term equals 0. For example, the contribution from the right leaf in the conditional probability tree for X_2 is $0 + 1 \log 1 - \frac{2}{2} \log 3 = 0 - \log 3$. The intuition is that the higher the BIC score, the more accurately we can predict the value of X_i given the values of its parents in G .

3 Markov Decision Processes

A finite Markov decision process (MDP) is a tuple $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$, where S is a finite set of states, A is a finite set of actions, $\Psi \subseteq S \times A$ is a set of admissible state-action pairs, P is a transition probability function, and R is an expected reward function. As a result of executing action $a \in A_s \equiv \{a' \in A \mid (s, a') \in \Psi\}$ in state $s \in S$, the process transitions to state $s' \in S$ with probability $P(s' \mid s, a)$ and receives an expected reward $R(s, a)$. In the discounted case, a solution to an MDP is a stochastic policy π that, for each $t > 0$, maximizes the expected return $R_t = E\{\sum_{k=t}^{\infty} \gamma^{k-t} R(s^k, a^k)\}$, where $\gamma \in (0, 1]$ is a discount factor, by selecting action a^k with probability $\pi(s^k, a^k)$ in each state s^k .

A factored MDP is described by a set of state variables \mathbf{S} . We use the coffee task [2], in which a robot has to deliver coffee to its user, as an example of a factored MDP. The coffee task is described by six binary state variables: S_L , the robot's location (office or coffee shop); S_U , whether the robot has an umbrella; S_R , whether it is raining; S_W , whether the robot is wet; S_C , whether the robot has coffee; and S_H , whether the user has coffee. Let $\{i, \bar{i}\}$ be the values of state variable S_i , where \bar{L} is the office and L the coffee shop. An example state is $\mathbf{s} = (\bar{L}, U, R, \bar{W}, C, \bar{H})$. The robot has four actions: G0, causing its location to change and the robot to get wet if it is raining and it does not have an umbrella; BC (buy coffee) causing it to hold coffee if it is in the coffee shop; GU (get umbrella) causing it to hold an umbrella if it is in the office; and DC (deliver coffee) causing the user to hold coffee if the robot has coffee and is in the office. All actions have a chance of failing. The robot gets a reward of 0.9 when the user has coffee plus a reward of 0.1 when it is dry.

3.1 DBN Model of Factored MDPs

The DBN model of a factored MDP [2] contains one DBN for each action $a \in A$. Like the original model, we assume that the conditional probabilities of the DBNs are represented using trees as opposed to tables. This allows for a more compact representation of the conditional probabilities. Figure 2 shows the DBN for action G0 in the coffee task. Assuming action G0 is executed at time t , the DBN determines the resulting values of state variables at time $t + 1$. For each state variable S_i , there are two nodes in the DBN: one node S_i^t representing the value of S_i at time t , and one node S_i^{t+1} representing its value at time $t + 1$. The same is true for the expected reward R . The value of S_i at time $t + 1$ depends on the values of state variables that have edges to S_i^{t+1} in the DBN. A dashed line indicates that a state variable is unaffected by G0.

Figure 2 also illustrates the conditional probability tree associated with state variable S_W and action G0, which we denote $\mathcal{T}_W^{\text{G0}}$. At each leaf, the first value represents the

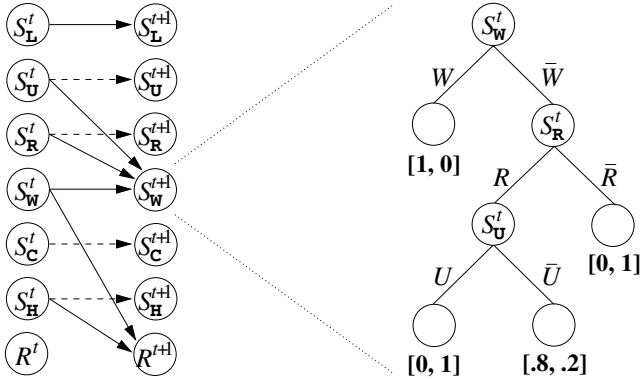


Fig. 2. The DBN for action G0 in the coffee task

probability that the robot is wet after executing G0, while the second value represents the probability that the robot is dry. At time t , if the robot is not wet (\bar{W}), it is raining (R), and the robot does not have an umbrella (\bar{U}), the conditional probability tree indicates that the robot is wet at time $t + 1$ with probability 0.8. We assume that there are no edges between state variables at a same time step. The transition probabilities are given by $P(\mathbf{s}^{t+1} | \mathbf{s}^t, a) = \prod_i P_a(S_i^{t+1} = f_{\{S_i^{t+1}\}}(\mathbf{s}^{t+1}) | \mathbf{Pa}(S_i^{t+1}) = f_{\mathbf{Pa}(S_i^{t+1})}(\mathbf{s}^t))$, where P_a is the joint probability distribution represented by the DBN for action a .

4 Learning a DBN Model

We develop an algorithm for learning DBN models of factored MDPs through interaction with the environment. Our algorithm builds a tree T_i^a for each pair of a state variable S_i and action a , approximating the conditional probabilities of S_i^{t+1} as a result of executing a . The family of trees for a implicitly defines the DBN for a . There is an edge between state variables S_j^t and S_i^{t+1} in the DBN if at least one node in T_i^a distinguishes between values of S_j^t . To build the tree T_i^a , the algorithm starts with a small tree and collects data by executing actions in the environment. Each time action a is executed, the algorithm records a data instance consisting of the former state, the resulting state, and the reward received. Each data instance maps to exactly one leaf of T_i^a , at which it is stored. We say that a leaf is empty if its corresponding set of data instances is empty.

A refinement at a leaf distinguishes between values of a state variable S_j^t and introduces a new leaf of T_i^a for each value of S_j^t . S_j^t is only considered for refinement if no internal nodes on the path from the root to the leaf of T_i^a already distinguish between values of S_j^t . As we have already mentioned, the BIC and BDe scores decompose into a local score for each leaf. Our algorithm evaluates a refinement by comparing the total score of the new leaves with the score of the old leaf. If at least one refinement increases the overall score, the algorithm retains the refinement that results in the largest increase. Regardless of the outcome, data instances at the old leaf are discarded. Our

approach is more sophisticated than adding edges in the graph of the DBN, since trees store conditional probabilities more compactly than tables.

Evaluating a refinement using the BIC and BDe scores really amounts to performing a statistical test to compare the posterior probabilities of two Bayesian networks given the data. It is well known that the accuracy of statistical tests, such as Chi-square, depends on having enough examples in each bin. At each leaf, and for each potential split variable S_j^t , the algorithm maintains a distribution vector M . Each entry M_k of the vector indicates the number of data instances at the leaf that assign the value k to S_j^t . When the algorithm evaluates a refinement over S_j^t , the distribution vector M determines how the data instances at the leaf will be distributed to the new leaves of T_i^a . We define a threshold parameter K and let our algorithm evaluate a refinement as soon as at least K data instances map to each non-empty leaf for each split variable.

In some tasks, the BIC and BDe scores fail to detect most of the refinements necessary to learn an accurate DBN model. The BIC score in Equation (1) is composed of a log likelihood term, which measures the likelihood of the data given a network, and a penalty term, which penalizes a network for having many parameters. The penalty term causes the BIC score to be less sensitive to improvements to the log likelihood since each refinement increases the number of parameters. We use regularization (14) to address this issue. In regularization, a functional is defined as the sum of a fidelity term and a stabilizer term. The stabilizer term is weighted by a parameter λ . We can multiply the penalty term of the BIC score by a parameter λ to put it in the form of a fidelity term and a stabilizer term:

$$\log[P(D | B)P(B)] \approx L(D | B) - \lambda \frac{|\theta|}{2} \log |D|, \quad (3)$$

such that λ controls the magnitude of the penalty for having many parameters.

4.1 Active Learning

Efficient data collection should gather sufficient data as quickly as possible. Since our algorithm requires at least K data instances to map to each non-empty leaf, the distribution of data instances across potential new leaves should be as uniform as possible for each possible refinement. The more skewed the distribution, the longer it takes to collect sufficient data to evaluate refinements. We devise the following scheme for active learning of DBNs. Before executing action a , the current state determines which leaf of T_i^a the resulting data instance will map to. When deciding which action to execute, we look at how the distribution vectors at corresponding leaves would change as a result of executing each action. To evaluate the change, we compute the entropy $H(M)$ of each distribution vector M :

$$H(M) = - \sum_k \theta_k \log \theta_k,$$

where $\theta_k = M_k / \sum_j M_j$. $H(M)$ is a non-negative function which is maximized when all entries of M are equal. An increase in $H(M)$ means that the distribution is becoming more uniform; a decrease means that it is becoming more skewed. The change in $H(M)$ can be computed in constant time. In each state, the active learning scheme selects the

action with largest total increase in the value of $H(M)$. With probability $\epsilon \in [0, 1]$, or if no action results in an increase of $H(M)$, the scheme selects a random action.

By maximizing the entropy, our active learning scheme maintains uniform distributions at the leaves, which in turn causes evaluation to occur as quickly as possible. However, the time it takes to collect data also depends on how often leaves are visited. The proposed scheme only implicitly affects the frequency with which leaves are visited, and assumes that each leaf is visited relatively frequently. Our approach is motivated by the fact that we want to use local information only to guide exploration. We believe that under this constraint, the entropy measure is best suited for the problem. By storing global information about the frequency with which leaves have been visited, it would be possible to try to steer exploration towards the least visited areas of the state space, although it is unclear how the system would know how to get to these areas.

5 Results

We ran experiments with our DBN learning approach in the coffee task [2], the Taxi task [17], and a simplified autonomous guided vehicle (AGV) task [18]. In each task, we compared our active learning scheme with passive learning, i.e., random action selection. In both cases, we used our approach for growing the conditional probability trees to implicitly learn the DBNs. Note that because of our assumption regarding data collection there is no meaningful way to compare our results to existing techniques for active learning. We had access to the true DBN model of each task and knew how many refinements of the trees were necessary to learn the true model. Figure 3 shows results of our experiments in the coffee task. The graph shows the number of correct refinements (out of 7) detected over time, averaged over 100 trials. Time is measured as the number of actions executed, not actual computer runtime. For each tree T_i^a , we used the parameter values $\epsilon = 0.3$, $K = 50|Dom(S_i)|$, where $Dom(S_i)$ is the domain of the state variable S_i whose conditional probabilities T_i^a approximates. Note that active learning outperformed passive learning and that the BIC and BDe scores performed almost identically.

In the Taxi task [17], a taxi agent has to deliver passengers from a pick-up location to their destination. In this case, the BIC and BDe scores fail to detect most of the refinements necessary to learn the true DBN model. We tested our modification to the BIC score in Equation (3) to see if regularization can improve the accuracy of the learned DBN model. Figure 4 shows results of the experiments in the Taxi task, averaged over 25 trials. In the Taxi task, the true DBN model requires 21 refinements. We used $\epsilon = 0.6$, $K = 50|Dom(S_i)|$, and report results of the BIC score for $\lambda = 0.1$ and $\lambda = 1$. The BDe score performed identically to the BIC score for $\lambda = 1$. Note that active and passive learning using the original BIC score ($\lambda = 1$) failed to detect many of the refinements of the true DBN model. With $\lambda = 0.1$, active and passive learning detected all of the refinements, with the active learning scheme being faster. We tested for values of λ between 0 and 1 in increments of 0.05, and $\lambda = 0.1$ gave the best results empirically, although we did not perform any sensitivity analysis.

In the AGV task [18], an autonomous guided vehicle has to transport parts between machines in a manufacturing workshop. We simplified the task by reducing the number

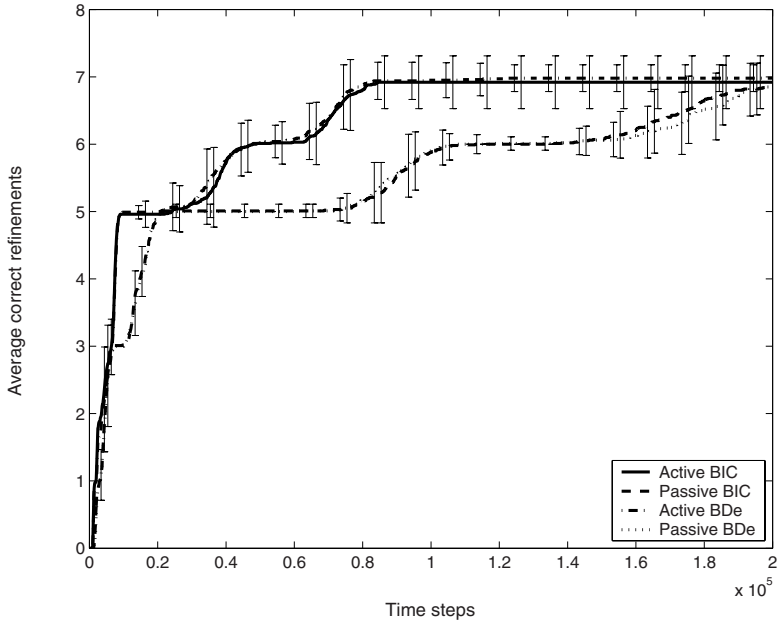


Fig. 3. Results in the coffee task

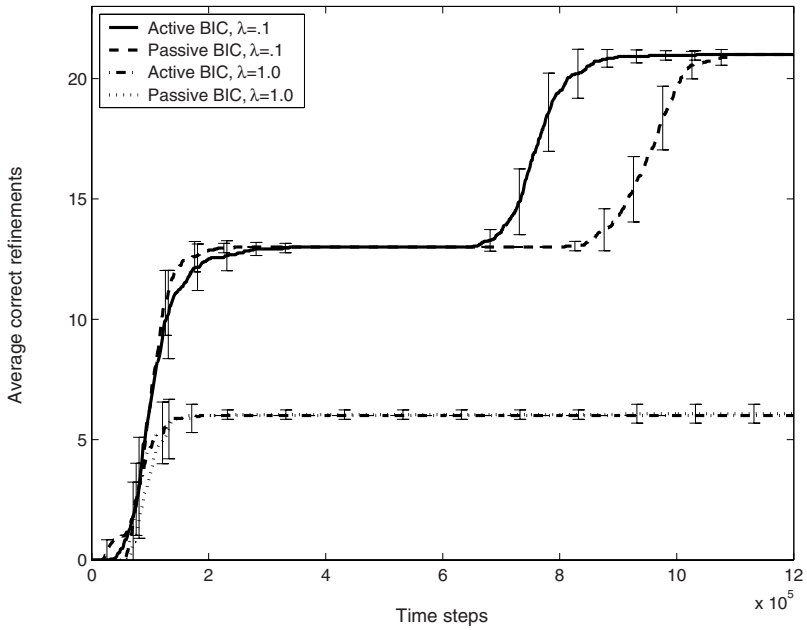


Fig. 4. Results in the Taxi task

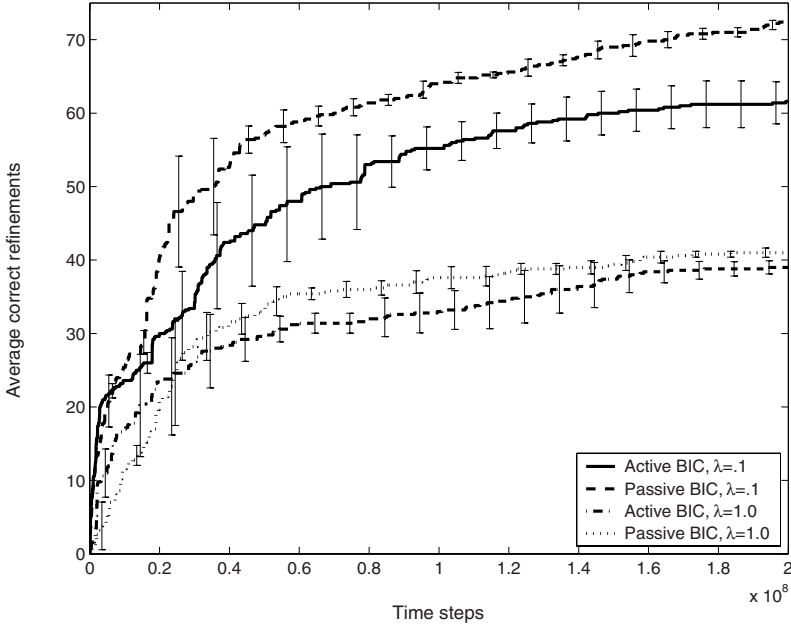


Fig. 5. Results in the AGV task

of machines to 2 and made it fully observable by setting the processing time of machines to 0. The resulting task has 75,000 states and 6 actions, and the true DBN model requires 162 refinements. Figure 5 shows results of the experiments in the AGV task, averaged over 5 trials. We used $\epsilon = 0.6$, $K = 50|Dom(S_i)|$, and report results of the BIC score for $\lambda = 0.1$ and $\lambda = 1$. There are several interesting things to notice. First, learning was very slow. We collected data for 200,000,000 time steps, and it is not clear that the graphs even converged. The learned DBN model did not come close to the true model, even for $\lambda = 0.1$. Also, passive learning actually outperformed active learning in the AGV task. We believe this is due to the fact that our active learning scheme selects actions based on local information, which we elaborate on in the conclusion. The results of the experiments in the AGV task indicate that learning DBN models of factored MDPs is a challenging problem, even using state-of-the-art metrics such as the BIC and BDe scores.

6 Conclusion

We have presented an algorithm for active learning of dynamic Bayesian networks in factored MDPs. Our approach is to learn DBNs by growing trees that represent the conditional probabilities of the DBNs. The algorithm stops to evaluate possible refinements of the trees as soon as a minimum number of data instances map to each relevant value of each potential split variable. To learn DBNs quickly, the distributions of data instances over values of each potential split variable should be uniform. We developed an

active learning scheme that selects actions with the goal of maintaining the distributions as uniform as possible.

Our active learning scheme selects actions based on local information, i.e., how the distributions change locally as a result of executing actions. This works well in tasks with limited size when all states are visited relatively frequently. However, in large tasks our scheme may fail to explore large regions of the state space, preferring to maintain uniformity in the current region. We believe this accounts for the results in the AGV task. To ensure that most or all of the state space is visited it is necessary to select actions based on global information. If global information is stored using trees its size is proportional to the number of leaves of the trees, not to the number of states, facilitating scaling. Reaching a specific region of the state space is difficult when we can only sample the current trajectory since we may not know which actions will get us there. Temporally-extended actions may provide a useful tool to achieve this. Although our work is an important first step, it needs to combine with further research to achieve accurate learning of DBNs in factored MDPs.

Acknowledgements. This work was partially funded by NSF grants ECS-0218125 and CCF-0432143. Any opinions, findings, 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. Dean, T., Kanazawa, K.: A model for reasoning about persistence and causation. *Computational Intelligence* 5(3), 142–150 (1989)
2. Boutilier, C., Dearden, R., Goldszmidt, M.: Exploiting structure in policy construction. In: *Proceedings of the International Joint Conference on Artificial Intelligence*. vol. 14, pp. 1104–1113 (1995)
3. Feng, Z., Hansen, E., Zilberstein, Z.: Symbolic Generalization for On-line Planning. In: *Proceedings of Uncertainty in Artificial Intelligence*. vol. 19, pp. 209–216 (2003)
4. Guestrin, C., Koller, D., Parr, R.: Max-norm Projections for Factored MDPs. In: *Proceedings of the International Joint Conference on Artificial Intelligence*. vol. 17, pp. 673–680 (2001)
5. Jonsson, A., Barto, A.: Causal Graph Based Decomposition of Factored MDPs. *Journal of Machine Learning Research* 7, 2259–2301 (2006)
6. Kearns, M., Koller, D.: Efficient Reinforcement Learning in Factored MDPs. In: *Proceedings of the International Joint Conference on Artificial Intelligence*. vol. 16, pp. 740–747 (1999)
7. Buntine, W.: Theory refinement on Bayesian networks. In: *Proceedings of Uncertainty in Artificial Intelligence*. vol. 7, pp. 52–60 (1991)
8. Friedman, N., Murphy, K., Russell, S.: Learning the structure of dynamic probabilistic networks. In: *Proceedings of Uncertainty in Artificial Intelligence*. vol. 14, pp. 139–147 (1998)
9. Heckerman, D., Geiger, D., Chickering, D.: Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning* 20, 197–243 (1995)
10. Murphy, K.: Active learning of causal Bayes net structure. Technical report, Computer Science Division, University of Berkeley (2001)
11. Steck, H., Jaakkola, T.: Unsupervised active learning in large domains. In: *Proceedings of Uncertainty in Artificial Intelligence*. vol. 18, pp. 469–476 (2002)

12. Tong, S., Koller, D.: Active learning for structure in Bayesian networks. In: Proceedings of the International Joint Conference on Artificial Intelligence. vol. 17, pp. 863–869 (2001)
13. Schwartz, G.: Estimating the dimension of a model. *Annals of Statistics* 6, 461–464 (1978)
14. Poggio, T., Girosi, F.: Regularization Algorithms for Learning that are Equivalent to Multi-layer Networks. *Science* 247, 978–982 (1990)
15. Sutton, R., Barto, A.: *Reinforcement Learning: An Introduction*. MIT Press, Cambridge (1998)
16. Chickering, D., Geiger, D., Heckerman, D.: Learning Bayesian networks: search methods and experimental results. In: *Proceedings of Artificial Intelligence and Statistics*. vol. 5, pp. 112–128 (1995)
17. Dietterich, T.: Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research* 13, 227–303 (2000)
18. Ghavamzadeh, M., Mahadevan, S.: Continuous-Time Hierarchical Reinforcement Learning. In: *Proceedings of the International Conference on Machine Learning*. vol. 18, pp. 186–193 (2001)

Boosting MUS Extraction

Santiago Macho González and Pedro Meseguer

IIIA, Institut d'Investigació en Intel·ligència Artificial
CSIC, Consejo Superior de Investigaciones Científicas
Campus UAB, 08193 Bellaterra, Catalonia, Spain
{smacho,pedro}@iiia.csic.es

Abstract. If a CSP instance has no solution, it contains a smaller unsolvable subproblem that makes unsolvable the whole problem. When solving such instance, instead of just returning the “no solution” message, it is of interest to return an unsolvable subproblem. The detection of such unsolvable subproblems has many applications: failure explanation, error diagnosis, planning, intelligent backtracking, etc. In this paper, we give a method for extracting a Minimal Unsolvable Subproblem (MUS) from a CSP based on a Forward Checking algorithm with Dynamic Variable Ordering (FC-DVO). We propose an approach that improves existing techniques using a two steps algorithm. In the first step, we detect an unsolvable subproblem selecting a set of constraints, while in the second step we refine this unsolvable subproblem until a MUS is obtained. We provide experimental results that show how our approach improves other approaches based on MAC-DVO algorithms.

1 Introduction

Constraint Satisfaction Problems (CSPs) have been applied with great success to tasks dealing with resource allocation, scheduling, planning, configuration and others. When a CSP instance has solution, the solver returns an assignment of values to variables such that all constraints are satisfied. But when a CSP instance is unsolvable, often the solver just returns a “no solution” message. In recent years, conflict-based reasoning is gaining interest in the field of constraint satisfaction. Instead of just certify that a CSP instance is unsolvable, more interesting is explaining why this instance has no solution. These explanations are useful in many settings: interactive applications, error diagnosis, planning, intelligent backtracking, etc. In early years, several authors focused on conflict based reasoning [19,8,22]. More recent work [13] focuses on extracting a minimal unsolvable subset (MUS) from the unsolvable problem, where minimal means that all subproblems of the MUS are solvable and all superproblems are unsolvable.

In the SAT field (Boolean satisfiability), many methods for finding MUSes have been developed. Early work [4,3,18,23] was limited to find a single unsatisfiable subformula (US) but without guaranteeing its minimality. These unsatisfiable subformula can be minimized into a MUS using the “Minimal Unsatisfiability Prover” developed in [11]. Very interesting is the work presented in [15] where authors developed a sound and complete technique for finding all MUSes of a CNF formula,

based on a strong relationship between maximal satisfiability and minimal unsatisfiability [17]. This relationship also was noted by [1].

The notion of *Maximal Satisfiable Subset* (MSS) as a complement of a MUS is presented in [16]. The authors show that MUSes and MSS are implicit encoding one of the other. They have shown that the complement of a MSS (CoMSS) is a hitting set of the set of MUSes and contains the minimal set of constraints that should be removed in order to restore consistency.

We have to mention the approach presented in [10] that computes a MUS using a two-step algorithm. Firstly they filter constraints that will not participate in the no-solution condition, obtaining an unsolvable subset of the original CSP. This subset is used in the second step, to identify the constraints that belong to a MUS. In order to have a competitive algorithm, authors use a solver that implements a MAC algorithm with dynamic variable ordering (DVO). Up to our knowledge, this combination achieves the highest efficiency among published approaches for MUS extraction.

The contribution that we present in this paper follows a similar strategy. Given a CSP instance without solution, in a first step we obtain an unsolvable subproblem by performing a forward checking search with dynamic variable ordering (FC-DVO), and computing the hitting set among the subsets of constraints involved in the no-solution condition. This process is iterated while getting unsatisfiable subproblems of lower size, with the help of a heuristic to select variables that are likely to be in a MUS. In a second step, once a MUS candidate has been selected, it is refined until obtaining a true MUS, as the second step of [10]. Experimentally, we obtain a significant improvement in performance with respect to the results of [10].

This paper is organized as follows. In Section 2, we discuss the relation of our approach with abstraction in constraint processing. In Section 3 we introduce the theoretical background needed for the paper. The detailed algorithm appears in Section 4, while the experimental results are in Section 5. There, we compare the performance of our approach against the algorithm described in [10] that is the most efficient implementation we have found to calculate a MUS. Finally in Section 6, we summarize our approach and discuss future work.

2 MUS and Abstraction

In the constraint reasoning literature, a new contribution to CSP solving is usually given at *low level*: typically, a new algorithm, heuristic, or combination of solving methods is presented in every detail, showing how the individual elements of a CSP instance (variables, values, constraints) evolve to find a solution or to show that none exists. On the other hand, contributions that see a CSP instance as a collection of subproblems that interact among them are much more scarce. We call these *high level* descriptions, where the emphasis is not on the atomic components of the instance, but on subproblems, as an intermediate entity between individual elements and the whole instance. High level descriptions provide an alternative view on CSPs, allowing for a kind of reasoning different

from the one based on atomic elements. For instance, one may want to find a subproblem having a particular property. This may generate heuristics of variable ordering, original ways of constraint processing or unexpected solving strategies. Although infrequent, this approach is not new in the constraint literature. Without trying to be exhaustive, we mention as representative examples the following works [6] [21] [5].

A simple example of high level description is the analysis of unsolvable CSP instances. If an instance has no solution, it contains at least one minimal unsatisfiable subproblem (MUS). Until this MUS is not solved (modifying it by removing some constraints or enlarging domains), the whole instance will remain without solution. Therefore, once we have seen that the original instance has no solution, the identification and extraction of this MUS by efficient algorithms is a primary goal, if we want to be able to solve the original CSP instance (in fact, an instance closer to the original instance, since this one is unsolvable).

High level descriptions can be seen as abstractions, where the emphasis is put on subproblem properties and particular details of atomic elements are ignored. Abstractions provide new perspectives on the problem, and allow for useful reasoning mechanisms. By no means we are advocating to consider reasoning at high level only, and forgetting the low level description. We stress the usefulness of reasoning at high level, but once it is done, you have to go down the low level and perform the work there. In some sense, reasoning at high level drives the computational activity to be performed at low level.

This paper combines reasoning at both levels. Our goal is to find an efficient way to extract a MUS, once the original instance has been proven unsolvable. Efficiency is crucial here, because after MUS extraction, the new instance has to be solved again. Without an efficient MUS extraction, the whole approach would be practically unfeasible. Reasoning at high level, we have devised an heuristic for selecting candidate variables for an hypothetical MUS. This heuristic is applied at low level, combined with a forward checking algorithm [9]. We obtain a unsolvable subproblem, which is later refined to obtain a true MUS, using an already known approach. Experimentally, we have seen that this heuristic gives quite good results, improving the efficiency of the most performant approach published up to date [10].

3 Theoretical Background

This Section provides the reader with the notions needed to follow the paper.

Definition 1 (CSP). *A CSP is defined by a tuple $\langle X, D, C \rangle$ where,*

- $X = \{x_1, x_2, \dots, x_n\}$ is a set of n variables.
- $D = \{D_1, D_2, \dots, D_n\}$ is a set of n domains, where variable x_k takes values in D_k .
- $C = \{c_1, c_2, \dots, c_r\}$ is a set of r constraints. A constraint c involves a sequence of variables $\text{var}(c) = \langle x_p, \dots, x_q \rangle$ denominated its scope. The extension of c is the relation $\text{rel}(c)$ defined on $\text{var}(c)$, formed by the permitted value tuples on the constraint scope.

A solution of the CSP is an instantiation of values to all variables such that the assigned values belong to the corresponding domains, and this instantiation satisfies all constraints in C . Sometimes, it is not possible to find such instantiation, in that case the problem is unsolvable.

Definition 2 (Subproblem). *Let $P = \langle X, D, C \rangle$ be a CSP. A subset of variables $S \subset X$ defines the subproblem $P|_S = \langle S, D|_S, C|_S \rangle$, where $D|_S$ is the subset of domains of variables in S and $C|_S$ is the subset of constraints with their scopes in S . The size of the subproblem is $|S|$.*

When a CSP is unsolvable, instead of return the message of “no solution”, could be interesting to return the unsolvable subproblem that makes unsolvable the whole problem. If we refine this unsolvable subproblem, identifying the minimal subset of constraints causing that the problem has no solution, we obtain a subproblem useful in many applications: explanation, diagnosis, planning, etc.

Definition 3 (Minimal Unsolvable Subproblem). *Let $P = \langle X, D, C \rangle$ be a CSP without solution. A minimal unsolvable subproblem is determined by a subset of variables $S \subset X$ such that $P|_S$ is unsolvable, but for any proper subset $S' \subsetneq S$, $P|_{S'}$ is solvable.*

Definition 4 (Hitting Set). *Given a collection of sets $S = \{S_1, \dots, S_n\}$, a hitting set of S , $HST(S)$, is a set that contains at least one element from each set S_1, \dots, S_n , that is, $\forall S_i \in S, HST(S) \cap S_i \neq \emptyset$.*

Example 1. Let $S = \{S_1, S_2, S_3\}$ where $S_1 = \{c_{12}\}$ $S_2 = \{c_{03}, c_{23}, c_{13}\}$ $S_3 = \{c_{23}, c_{13}\}$. There are several hitting sets of S , i.e: $HST_1(S) = \{c_{12}, c_{23}\}$ $HST_2(S) = \{c_{12}, c_{13}\}$ $HST_3(S) = \{c_{12}, c_{13}, c_{03}\}$.

The hitting set problem can be prove to be NP-complete by a reduction from the vertex cover problem [7].

Proposition 1. *Let $P = \langle X, D, C \rangle$ be a CSP without solution, explored by the forward checking (FC) algorithm. Let $CONS \subset X$ be the subset of variables that have been assigned by FC. Let $EMPTY \subset X$ be the subset of variables for which an empty domain has been detected. Calling $S = CONS \cup EMPTY$, the subproblem $Q = \langle S, D|_S, C' \rangle$, where $C' = \{c \in C | c \text{ is responsible for eliminating values of the domain of a variable that either was assigned or became empty at each branch}\}$ is unsolvable.*

Proof. To prove this result, it is enough to realize that FC only assigns variables in $CONS$ and only requires variables in $EMPTY$ to detect that there is no solution, using constraints of C' . Therefore, if $S = CONS \cup EMPTY$, FC will find that Q has no solution, by simply repeating the variable instantiation order used in the FC execution on P . □

Thus, following proposition [1], we can obtain an equivalent unsolvable subproblem of the original unsolvable CSP using a FC Solver. It is important to remark that proposition [1] is true either for static or dynamic variable ordering.

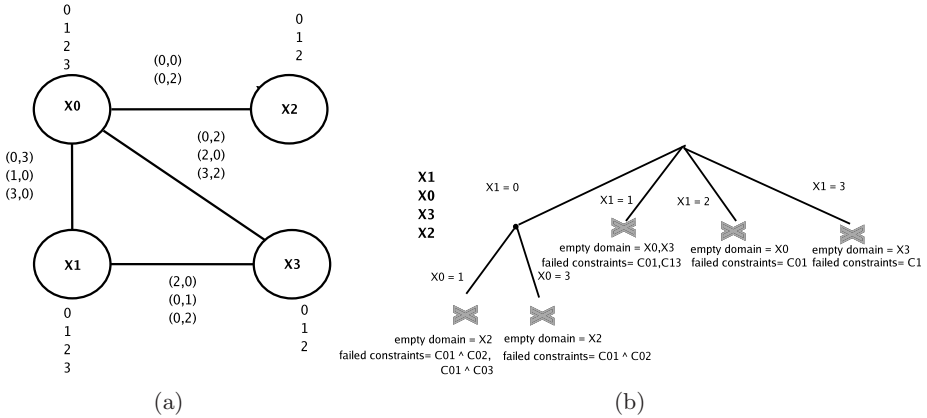


Fig. 1. (a) CSP (b) Its search tree generated by the FC algorithm

Example 2. Let $MUS = \{x_0, x_1, x_2\}$ be a minimal unsolvable subset of the CSP shown in Figure 1(a) where constraints indicate allowed values between variables. Figure 1(b) shows the corresponding search tree generated by FC algorithm with static order x_1, x_0, x_3, x_2 . Here $CONS = \{x_1, x_0\}$, $EMPTY = \{x_2, x_0, x_3\}$.

Example 3. Figure 2 shows the generated unsolvable subproblem following proposition 1. This subproblem is made by the union of the failed constraints of all branches. In the example, $C = \{c_{01} \wedge c_{02}, c_{01} \wedge c_{03}, c_{01} \wedge c_{02}, c_{01}, c_{13}, c_{01}, c_{13}\}$ that is equivalent to $C = \{c_{01}, c_{02}, c_{03}, c_{13}\}$. In this example, the unsolvable subproblem obtained is equal than the original CSP.

We notice in example 3 that the selected constraints are more than enough to guarantee the no-solution condition. If we study in more detail the search tree of the Figure 1(b), in every failed branch we have a disjunction of constraints, i.e:

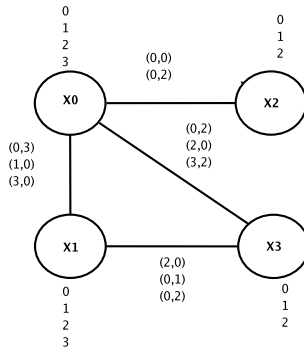


Fig. 2. An unsolvable subproblem of the CSP shown in Figure 1(a)

the leftmost branch has as failed constraints $\{c_{01} \wedge c_{02}, c_{01} \wedge c_{03}\}$. That means that in this branch, only the constraint $c_{01} \wedge c_{02}$ or the constraint $c_{01} \wedge c_{03}$ is necessary to produce an empty domain in variable x_2 . The same analysis is valid for all branches of the search tree. Thus, if we select at least one constraint from each branch, the resulting set will be an unsatisfiable subset of the original problem. This is the definition of *Hitting Set*. The set of constraints of every branch, CC_i , is made by selecting the constraints C' that justify failure in each branch as explained in proposition 1. This generates the following result.

Proposition 2. *Let $\langle X, D, C \rangle$ be a CSP without solution, explored by FC. Let CC be the collection of subsets of constraints that justify failure at each branch. Then, $\langle X, D, HTS(CC) \rangle$ is unsolvable.*

Proof. $CC = \{CC_1, \dots, CC_k\}$ is the collection of subsets of constraints justifying failure, one for each branch. The structure of one of these subsets is $CC_i = \{\{c_{i_1}, \dots, c_{i_p}\}, \dots, \{c_{i_r}, \dots, c_{i_k}\}\}$, meaning that each element of CC_i is enough to justify failure in its branch. Since $HTS(CC)$ takes at least one element of each subset CC_i , FC on the subproblem $\langle X, D, HTS(CC) \rangle$ will also fail in every branch. So this subproblem has no solution. \square

Example 4. In the CSP of the Figure 1(a), let CC_i represents the set of the selected failed constraints by proposition 1. From the leftmost branch to the rightmost branch of the search tree 1(b), we obtain: $CC_1 = \{c_{01} \wedge c_{02}, c_{01} \wedge c_{03}\}$, $CC_2 = \{c_{01} \wedge c_{02}\}$, $CC_3 = \{c_{01}, c_{13}\}$, $CC_4 = \{c_{01}\}$, $CC_5 = \{c_{13}\}$. Let $CC = \{CC_1, CC_2, CC_3, CC_4, CC_5\}$. Every *Hitting Set* of CC produces an unsolvable subset. i.e: $HST(CC) = \{c_{01}, c_{02}, c_{13}\}$ as shown in Figure 3.

Thus, we can develop an algorithm to obtain an unsolvable subset, firstly selecting the constraints that justify failure at every branch of the search tree and afterwards calculating its HST. In addition, if the variables that are likely to be in a MUS are put first in the search tree, the subproblem size tend to be smaller. The following result helps to detect such variables.

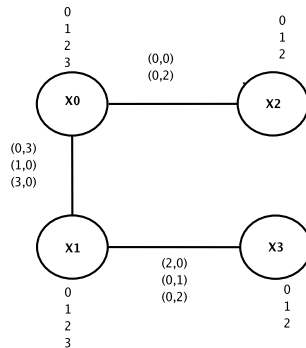


Fig. 3. An unsolvable subproblem of the CSP shown in Figure 1(a)

Proposition 3. *Let a CSP without solution, explored by FC-SVO. Let $EMPTY \subset X$ be the subset of variables for which an empty domain has been detected during search. $EMPTY$ contains a variable of a MUS.*

Proof. Let us consider the deepest branch instantiated by FC-SVO, formed by the ordered set of variables $\{x_1, \dots, x_k\}$. From the FC-SVO search, we know that $\{x_1, \dots, x_k\} \cup EMPTY$ is unsolvable. But the subset of variables $\{x_1, \dots, x_k\}$ is solvable, because FC-SVO has instantiated it. Therefore, it should exist at least one variable in $EMPTY$, the other subset of variables, that makes the whole subproblem unsolvable. To see that it must belong to a minimal unsolvable subproblem, it is enough to realize that inside any unsolvable subproblem there is a minimal unsolvable one. \square

We will use this result, valid for static variable ordering only, as a heuristic when forward checking uses dynamic variable ordering. This heuristic will help trying to detect MUS variables, as we will see in the next Section.

4 The CORE-FC Algorithm

In this Section we present the algorithm **CORE-FC**, that implements our approach. It extracts a MUS from an unsolvable CSP instance, using the theoretical background previously introduced. Having an unsolvable CSP, the basic idea of **CORE-FC** (algorithm 1) is first to generate an unsolvable subset using the function **CORE-FC1** and afterwards refine this subset with function **CORE-FC2** until a minimal subset is found. We describe these algorithms in the following.

4.1 The CORE-FC1 Algorithm

As mentioned before, the goal of **CORE-FC1** (algorithm 2) is to obtain an unsolvable subproblem (not necessary minimal) of an unsolvable CSP instance. This algorithm is based on propositions 2 and 3. We decided to use a solver based on a Forward Checking algorithm with Dynamic Variable Ordering (FC-DVO), where variables are selected according to the popular minimum domain heuristic 9. The reason for this choice is that using a solver based on a backtracking with DVO is not competitive. A solver based on MAC-DVO is competitive, but we notice that we can select a smaller subset of constraints involved in a MUS by using a solver based on FC-DVO. Experiments show that FC-DVO tends to

Algorithm 1. The CORE-FC algorithm

Require: X, D, C is an unsolvable problem

1: **procedure** **CORE-FC**(X, D, C)

2: $X_{us}, D_{us}, C_{us} \leftarrow \text{CORE-FC1}(X, D, C)$ $\{X_{us}, D_{us}, C_{us}$ is an unsolvable subset of $X, D, C\}$

3: $X_{mus}, D_{mus}, C_{mus} \leftarrow \text{CORE-FC2}(X_{us}, D_{us}, C_{us})$

4: **return** $X_{mus}, D_{mus}, C_{mus}$

Ensure: $X_{mus}, D_{mus}, C_{mus}$ is a minimal unsolvable subset of X, D, C

find smaller MUS candidates than MAC-DVO. We explain this fact as follows. MAC performs arc-consistency on every constraint. When MAC realizes that the instance has no solution, it has to include each constraint that is responsible for removing a value into the MUS candidate (this includes constraints among variables that never have been instantiated). On the other hand, FC propagation is simpler: it performs arc consistency on constraints connecting assigned and unassigned variables at any stage of search. In particular, constraints among variables that have never been instantiated are not considered. For this reason, we believe that FC focuses better than MAC on suitable MUS candidates. This intuition is confirmed in practice (see Section 5).

CORE-FC1 works as follows. It takes as input an unsolvable CSP instance. It returns as output an unsolvable subproblem of that instance. Firstly, it pass the instance to a modified FC-DVO solver. This solver takes as input a CSP instance and a variable (line 5). This variable is forced to be the first one instantiated by the solver although the DVO criteria does not select it first. The reason for this will become apparent later. As output, it returns a variable with empty domain in the deepest branch explored by FC-DVO.

In CC we collect the subsets of constraints that justify failure at each branch of FC-DVO (line 6). By proposition 2, we know that the subproblem generated by the hitting set of CC , $HST(CC)$, has no solution. Therefore, we replace the input instance by this subproblem. Algorithm 3 computes the Hitting Set of a set of constraints. This function is used by the algorithm 2 in order to obtain an unsolvable subproblem (line 7).

To decrement the size of the unsolvable subproblem, one can repeat the above process with a different variable ordering. If variables that belong to a MUS are instantiated at the first levels of the FC search tree, the resulting unsolvable subproblem tend to be smaller. Following this idea, we consider that a variable with empty domain in the deepest branch explored by FC-DVO in the current iteration it is likely to be in a MUS. This is based on proposition 3, which expresses a result valid for SVO, while we use it here as heuristic for DVO. Then, we take that variable to be instantiated first at the next iteration (line 8). This variable is returned by the FC-DVO solver. The whole process iterates until the unsolvable subproblem does no longer decrement its size (line 9).

Algorithm 2. The CORE-FC1 algorithm

Require: X, D, C is an unsolvable problem

```

1: function CORE-FC1( $X, D, C$ )
2:    $us_{cur} \leftarrow C$ 
3:   repeat
4:      $us_{prev} \leftarrow us_{cur}$ 
5:      $x_k \leftarrow \text{solveCSP-FC-DVO}(X, D, C, x_1)$  {  $x_k$  is candidate to belong to the MUS}
6:      $CC \leftarrow$  collection of subsets of constraints that justify failure at each branch of FC-DVO
7:      $us_{cur} \leftarrow HST(CC)$ 
8:     reorder variables so that  $x_k$  becomes  $x_1$ 
9:   until  $|us_{prev}| \leq |us_{cur}|$ 
10:   $\langle X_{us}, D_{us}, C_{us} \rangle \leftarrow \text{generateCSP}(us_{prev})$ 
11:  return  $\langle X_{us}, D_{us}, C_{us} \rangle$ 

```

Ensure: $\langle X_{us}, D_{us}, C_{us} \rangle$ is an unsolvable subset of $\langle X, D, C \rangle$

Algorithm 3. *The Hitting Set algorithm*

```

1: procedure HST( $CC$ )
2:  $hittingSet \leftarrow \emptyset$ 
3:  $CC \leftarrow initialPurge(CC)$ 
4: while  $\neg isHittingSet(hittingSet)$  do
5:    $c \leftarrow chooseConstraint(CC)$ 
6:    $hittingSet \leftarrow hittingSet \cup \{c\}$ 
7:    $CC \leftarrow purge(CC)$ 
8: end while
9: return  $hittingSet$ 

```

We are interested in a hitting set algorithm that minimizes the number of variables that are added to the HST when a new constraint is included. Unfortunately [14] shows that calculating the minimal hitting set is an NP-hard problem. We implement a strategy to minimize the number of variables that are included in the hitting set using the function *chooseConstraint*. This function selects the constraint that minimizes the number of variables that we include in the HST. Note that this heuristic does not guarantee that the resulting hitting set has a minimum number of variables, but empirically it works well and provides good results. The *purge* function remove sets that are superset of others.

4.2 The CORE-FC2 Algorithm

CORE-FC2 (algorithm 4) refines the unsolvable instance that takes as input, and returns a MUS of that instance. It is based on the *dcMUC* function shown in [10]. The algorithm enters in a loop (lines 4 - 12) where new variables that belongs to the MUS are discovered using the *DichotomicSearch* function (line 5). When a new variable x_k is discovered, it is included in the MUS candidate (line 6). If this candidate has no solution (line 9), then it is confirmed as MUS (line 10) and the loop ends. Otherwise, the original instance is searched again for more variables in the MUS, looking into the subproblems that at least contain $\{x_0, \dots, x_{k+1}\}$ (line 12).

Algorithm 4. *The CORE-FC2 algorithm*

Require: X, D, C is a superset of a MUS

```

1: function CORE-FC2( $X, D, C$ )
2:  $X_{MUS} \leftarrow \emptyset, D_{MUS} \leftarrow \emptyset, C_{MUS} \leftarrow \emptyset$ 
3:  $MUS \leftarrow \text{false}, k \leftarrow 0$ 
4: while  $\neg MUS$  do
5:    $x_k \leftarrow DichotomicSearch(X, D, C, k)$ 
6:    $X_{MUS} \leftarrow X_{MUS} \cup \{x_k\}$  {we include this var to the MUS}
7:    $C_{MUS} \leftarrow C_{MUS} \cup \{c_{ik}\}$  { $c_{ik}$  is a set of constraints involving variable  $x_k$ }
8:    $D_{MUS} \leftarrow D_{MUS} \cup \{d_k\}$ 
9:   if  $solveCSP-MAC-DVO(X_{MUS}, D_{MUS}, C_{MUS}) = UNSAT$  then
10:     $MUS \leftarrow \text{true}$ 
11:   end if
12:    $k \leftarrow k + 1$ 
13: end while
14: return  $X_{MUS}, D_{MUS}, C_{MUS}$  {minimal unsolvable subproblem}

```

Ensure: $X_{MUS}, D_{MUS}, C_{MUS}$ is a MUS

Algorithm 5. *The Dichotomic Search algorithm*

```

1: procedure DichotomicSearch( $X, D, C, k$ )
2:  $min \leftarrow k, max \leftarrow |X|$ 
3: while  $min \neq max$  do
4:    $center \leftarrow (min + max)/2$ 
5:    $X_{DIC} \leftarrow \{x_0, \dots, x_{center}\}$ 
6:    $C_{DIC} \leftarrow$  set of constraints involving vars  $\{x_0, \dots, x_{center}\}$ 
7:    $D_{DIC} \leftarrow \{d_0, \dots, d_{center}\}$ 
8:   if solveCSP-MAC-DVO( $X_{DIC}, D_{DIC}, C_{DIC}$ ) = SAT then
9:      $min \leftarrow center + 1$ 
10:  else
11:     $max \leftarrow center$ 
12:  end if
13: end while
14: return  $x_{min}$ 

```

In order to identify a variable that belongs to a MUS, the next procedure can be used. Starting from the first variable in the given order, add at every step one more variable until the CSP becomes unsatisfiable. When that occurs the last added variable belongs to a MUS. If we want to speed up this algorithm, we can use a dichotomic search. The *DichotomicSearch* function (algorithm 5) starts a dichotomic search in order to find a variable that belongs to the MUS (similar to function *dcTransition* described in [10]). It searches this variable in the set $\{x_{min}, \dots, x_{max}\}$. Initially, index *min* takes value *k*, while *max* takes the total number of variables. Parameter *k* is the limit between the discovered variables of the MUS and the undiscovered ones. The algorithm enters in a loop between lines 3-12 until a variable of the MUS is discovered. It takes the set $\{x_0, \dots, x_{center}\}$ and checks if it is solvable or not. If it is solvable, it searches for the variable of the MUS among the set $\{x_{center+1}, \dots, x_{max}\}$. If the problem is unsolvable then the variable belongs to the set $\{x_{min}, \dots, x_{center}\}$. Doing this procedure, the variable that belongs to the MUS is obtained when $x_{max} = x_{min}$ (line 3).

5 Experimental Results

In this section we have performed several experiments in order to compare the performance of our approach against the **PCORE+WSCORE** algorithm described in [10], that at the present seems to be the most performant published algorithm. We use non competitive FC and MAC solvers based on the JCL library [12,20].

The **PCORE+WSCORE** described in [10] works as follows. This is a 2-step algorithm, the PCORE step and the WSCORE step. In the PCORE step, a MAC-DVO solver is used to return an unsolvable subproblem made by all the constraints that during the search removed at least one value in the domain of a variable. A dom/wdeg heuristic is used to choose the order in which variables will be instantiated. Calls to the MAC-DVO solver are done (updating the heuristic at every call) until the size of the obtained subproblem does not longer decrease. Once the PCORE step returns an unsolvable subproblem (not minimal), the WSCORE step extract a MUS from this unsolvable subproblem, using a dichotomic search.

Table 1. Results for the PCORE+WCORE algorithm

BENCHMARK			PCORE + WCORE					
Name	VARS	CONS	SIZE US PCORE	CHECKS PCORE	SIZE MUS WCORE	TIME	CHECKS	VIS NODES
randomB-25-58	25	58	19	4181	8	9.3s	60731	569
randomB-16-90	16	90	16	2648	7	4.2s	25072	364
randomB-26-6	26	63	17	7591	6	3.7s	17534	228
randomB-31-347	31	347	20	2550	8	9.8s	55033	582
randomB-43-176	43	176	35	34859	15	50.7s	381684	2012
randomB-36-470	36	470	21	1116	7	5.8s	10719	314
randomB-48-220	48	220	29	39116	16	47.1s	410452	1992
randomB-45-739	45	739	19	1818	7	8.3s	13725	426
pigeons5	5	10	5	1400	5	0.23s	3903	250
pigeons6	6	15	6	8250	6	0.76s	19683	883
pigeons7	7	21	7	52092	7	3.73s	102378	4219
pigeons8	8	28	8	369446	8	25.22s	618219	26825
pigeons9	9	36	9	2963760	9	219.54s	4597144	209178
pigeons10	10	55	10	26686962	10	2458s	40353999	1872587
pigeons11	11	55	11	266889620	11	26324s	400961870	18708727
dual-ehi-85-297-0	297	4094	60	120167	25	742.98s	1489447	10001
dual-ehi-85-297-1	297	4112	83	143760	19	738.69s	988227	7136
dual-ehi-85-297-7	297	4111	82	152272	18	627.45s	1167588	6030
dual-ehi-85-297-9	297	4118	64	55215	20	599.06s	911398	6322
dual-ehi-85-297-18	297	4120	69	144520	18	683.49s	927860	6577
dual-ehi-85-297-20	297	4106	72	85655	20	576.08s	1033272	6166
dual-ehi-85-297-24	297	4105	70	89563	19	694.70s	1037183	6703
dual-ehi-85-297-26	297	4102	19	115150	19	282.94s	1032841	6876
dual-ehi-85-297-27	297	4120	57	80516	20	575.42s	939922	6399
dual-ehi-85-297-44	297	4130	70	96148	16	474.16s	639734	4677
dual-ehi-85-297-49	297	4124	61	118023	22	495.73s	1255883	7603
dual-ehi-85-297-65	297	4116	74	147270	21	539.22s	975204	7997
dual-ehi-85-297-83	297	4099	90	169078	20	802.28s	1789755	8064
dual-ehi-85-297-88	297	4119	69	114815	18	603.50s	758366	6306
dual-ehi-85-297-92	297	4106	65	142880	20	595.57s	947950	7133
dual-ehi-85-297-99	297	4115	117	124209	19	710.32s	1319022	7371

We ran three different set of benchmarks. Firstly, we have generated unsolvable subproblems using a modified *random model B generator*, where we forced the random generator to return unsolvable CSPs. The second set of constraints is the well known problem of the *pigeons*, where we have to put n pigeons into n-1 boxes, one pigeon per box. These pigeons problems are interesting, because the whole problem is a MUS. Finally we ran experiments on the *dual-ehi* benchmarks that are 3-SAT instances converted to binary CSP instances using the dual method. The pigeons and the dual-ehi benchmarks can be found in [2].

Table 1 shows the performance of the algorithm proposed in [10], while table 2 shows the performance of our approach described in the previous section. The columns indicates: the name of the benchmark, the number of variables and constraints the benchmark has, the size of the US (not minimal) and the number of checks after the first step, the size of the obtained MUS, the execution time in seconds, the total number of checks and the number of visited nodes.

We study separately the results for the three different types of benchmarks.

Table 2. Results for the CORE-FC1+CORE-FC2 algorithm

BENCHMARK			CORE-FC1 + CORE-FC2					
Name	VARS	CONS	SIZE US CORE- FC1	CHECKS CORE- FC1	SIZE MUS CORE- FC2	TIME	CHECKS	VIS NODES
randomB-25-58	25	58	11	4452	7	1.32s	10931	391
randomB-16-90	16	90	8	2602	8	1.05s	12355	359
randomB-26-63	26	63	9	1176	8	1.19s	15498	271
randomB-31-347	31	347	8	5803	7	1.53s	9056	248
randomB-43-176	43	176	12	7705	10	4.40s	41842	974
randomB-36-470	36	470	7	2592	7	1.54s	3987	112
randomB-48-220	48	220	6	2954	5	1.78s	6110	230
randomB-45-739	45	739	7	3645	7	2.58s	5418	138
pigeons5	5	10	5	584	5	0.24s	3041	298
pigeons6	6	15	6	3170	6	0.79s	14489	1123
pigeons7	7	21	7	19452	7	4.45s	69506	5659
pigeons8	8	28	8	136850	8	36.75s	385201	36905
pigeons9	9	36	9	1095824	9	381.78s	2728478	289818
pigeons10	10	55	10	9863874	10	4161.96s	23529833	2598347
pigeons11	11	55	11	98640740	11	47335.87s	232711461	25966327
dual-ehi-85-297-0	297	4094	25	42881	25	252.39s	555210	7461
dual-ehi-85-297-1	297	4112	19	31790	19	227.96s	224513	4164
dual-ehi-85-297-7	297	4111	18	22259	18	236.66s	163585	3061
dual-ehi-85-297-9	297	4118	21	25756	20	187.65s	186456	3985
dual-ehi-85-297-18	297	4120	18	20782	18	187.22s	207200	3165
dual-ehi-85-297-20	297	4106	20	22432	20	187.95s	218315	3969
dual-ehi-85-297-24	297	4105	19	50346	19	248.93s	186793	5327
dual-ehi-85-297-26	297	4102	19	32803	19	241.15s	180389	4005
dual-ehi-85-297-27	297	4120	20	44817	20	198.32s	202829	5585
dual-ehi-85-297-44	297	4130	16	20296	16	232.46s	119877	1930
dual-ehi-85-297-49	297	4124	22	20971	22	248.95s	312244	4909
dual-ehi-85-297-65	297	4116	21	22402	21	245.80s	304262	4291
dual-ehi-85-297-83	297	4099	20	15077	20	189.53s	310092	3778
dual-ehi-85-297-88	297	4119	18	25994	18	186.60s	174942	3390
dual-ehi-85-297-92	297	4106	20	29802	20	189.48s	217860	4103
dual-ehi-85-297-99	297	4115	19	11820	19	185.97s	242962	3178

5.1 Random Benchmarks

We generated several unsolvable random problems using the model B. We have modified our generator in order to force it to return unsolvable instances. The generated problems have between 15 to 48 variables and from 58 to 739 constraints. It is important to point that we cannot control if the generated problems have more than one unsolvable subproblem, thus, it is possible that the algorithms find different MUSes. Comparing the benchmarks where both algorithms returns the same MUS, *randomB-36-470* and *randomB-45-739*, we notice that our first step returns smaller unsolvable subproblems than the first step of the PCORE+WCORE algorithm. The second step are equivalent for both algorithms, but our approach has the advantage that the unsolvable subproblem that is the input of the second step is smaller than the unsolvable subproblem of the PCORE+WCORE algorithm. Experiments show that our approach reduces the execution time, the number of checks and the number of visited nodes.

5.2 Pigeons Benchmarks

The pigeons benchmarks are problems where we have to put n pigeons into $n-1$ boxes, one pigeon per box. These problems have the characteristic that any proper subproblem is solvable (the whole problem is a MUS). Tables 1 and 2 show that with these benchmarks our algorithm has a worse performance in time than the PCORE+WCORE approach. We notice that the first step makes the difference; while our algorithm uses a FC-DVO solver, the PCORE+WCORE approach uses a MAC-DVO solver. In both approaches all constraints will be selected (the whole problem is unsolvable). The MAC-DVO solver is faster than the FC-DVO solver, thus there is an important gain in time at the end of the first step for the PCORE+WCORE algorithm over our approach. Our approach has a better number of checks because a FC solver does less checks than a MAC solver. In the opposite, the number of visited nodes is greater for the FC than the MAC due to the MAC propagation.

5.3 Dual-ehi Benchmarks

The dual-ehi benchmarks are problems where benchmarks that are 3-SAT instances are converted to binary CSP instances using the dual method. We ran several experiments with 297 variables and between 4094 to 4130 constraints. Tables 1 and 2 show that our algorithm has a better performance. In these benchmarks, we decrease the execution time by a factor of 3, also decreasing the number of checks and the number of visited nodes. It is interesting to point that very often the unsatisfiable subset obtained at the first step is a MUS. Therefore our second step is faster than the PCORE+WCORE approach, where the output of the first step is a bigger unsolvable subproblem.

6 Conclusions

We have developed a new approach for extracting a MUS from an unsolvable CSP instance. It is based on a two-step algorithm. In the first step, an unsolvable subproblem is obtained, using a FC-DVO solver combined with the computation of a hitting set on the constraints responsible for the no solution condition. To remove variables which do not belong to the minimal version of this subproblem, this process is iterated with the help of a heuristic to identify variables that are likely to be in a MUS. The iteration ends when the computed unsolvable subproblem does no longer decrement its size. The second step refines this unsolvable subproblem using a dichotomic search until a true MUS is found.

We compared our approach with the best approach we have found so far called PCORE + WCORE [10] which is also a two-step algorithm. The main difference between these two approaches occurs in the first step. While PCORE + WCORE iterates using a MAC-DVO solver, our approach iterates using a FC-DVO solver combined with the hitting set computation and the heuristic to select likely MUS variables. As result, our approach is able to compute MUS

candidates of smaller size (first step of **CORE-FC**), with less computational effort. As consequence, the effort required in the second step is also smaller. Experimental results show that our approach is beneficial in most benchmarks, although in some benchmarks (pigeons) our approach is not competitive. It is worth realizing that each pigeon instance is itself a MUS, so we hypothesize that when the original unsolvable instance is already minimal, our approach is not competitive (in that case, the hitting set computation and the heuristic do not bring any benefit, they add overhead only). But we believe that this is not the general case. Usually, unsolvable instances contain smaller MUSes, for which we believe that our approach is adequate. Our intuition behind the claim that a FC algorithm is better than a MAC algorithm for finding MUSes is that whilst a FC just considers constraints between past and future constraints a MAC algorithm tends to maintain the consistency of the CSP. Thus a MAC algorithm will select more candidate constraints than a FC algorithm. This explains why our algorithm finds a smaller and better unsolvable candidate.

The capacity of reasoning at subproblem level has been crucial to develop this approach. The hitting set idea considers the different subsets of constraints that are responsible for the no solution condition of the whole subproblem. The heuristic for variables likely to be in a MUS is inspired in a property of the complete subproblem. This view abstracts atomic CSP components, focusing on subproblems. We believe that this perspective offers new and interesting ways of reasoning in constraint solving, able to improve existing techniques or to develop new ones.

Acknowledgements

Authors thank anonymous reviewers for their constructive criticisms.

References

1. Bailey, J., Stuckey, P.J.: Discovery of Minimal Unsatisfiable Subsets of Constraints Using Hitting Set Dualization. In: Hermenegildo, M.V., Cabeza, D. (eds.) *Practical Aspects of Declarative Languages*. LNCS, vol. 3350, pp. 174–186. Springer, Heidelberg (2005)
2. Benchmark problems. <http://cpai.ucc.ie/05/Benchmarks.html>
3. Bruni, R.: Approximating minimal unsatisfiable subformulae by means of adaptive core search. *Discrete App. Math. Journal*, vol. 130, pp. 85–100. Elsevier Science Publishers, B.V. (2003)
4. Bruni, R., Sassano, A.: Restoring Satisfiability or Maintaining Unsatisfiability by Finding Small Unsatisfiable Subformulae. In: Bruni, R., Sassano, A. (eds.) *SAT 2001 (LICS) 2001 Workshop on Theory and Applications of Satisfiability Testing*, Boston, Massachusetts, USA, June 14-15, 2001, pp. 14–15. Elsevier Science Pub., North-Holland, Amsterdam (2001)
5. Faltings, B., Macho-Gonzalez, S.: Open Constraint Programming. *Artificial Intelligence*, vol 161, pp. 181–208 (2005)

6. Freuder, E., Hubbe, P.: Extracting Constraint Satisfaction Subproblems. In: Proc. of the 14th International Joint Conference on Artificial Intelligence. pp. 548–555 (1995)
7. Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York (1979)
8. Ginsberg, M.L.: Dynamic backtracking. *Journal of Artificial Intelligence Research* 1, 25–46 (1993)
9. Haralick, R., Elliot, G.: Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence* 14, 263–313 (1980)
10. Hemery, F., Lecoutre, C., Sais, L., Boussemart, F.: Extracting MUCs from constraint networks. In: Proceedings of the 17th European Conference on Artificial Intelligence (ECAI'06) (2006)
11. Huang, J.: MUP: a minimal unsatisfiability prover. In: Huang, J. (ed.) ASP-DAC '05. Proceedings of the 2005 conference on Asia South Pacific design automation, Shanghai, China, pp. 432–437. ACM Press, New York (2005)
12. Java Constraint Library (JCL): <http://liawww.epfl.ch/JCL/>
13. Junker, U.: QUICKXPLAIN: Conflict Detection for Arbitrary Constraint Propagation Algorithms. In: IJCAI'01 Workshop on Modelling and Solving problems with constraints (CONS-1) (2001)
14. Kar, R.M.: Reducibility among combinatorial problems. *Complexity of Computer Computations*, pp. 85–103 (1972)
15. Liffiton, M.H., Andraus, Z.S., Sakallah, K.A.: From Max-SAT to Min-UNSAT: Insights and Applications. Technical Report CSE-TR-506-05 (February 2005)
16. Liffiton, M.H., Moffitt, M.D., Pollack, M.E., Sakallah, K.A.: Identifying Conflicts in Overconstrained Temporal Problems. In: Proc. IJCAI-05, pp. 205–211, Edinburgh, Scotland (2005)
17. Liffiton, M.H., Sakallah, K.A.: On Finding All Minimally Unsatisfiable Subformulas. In: Proc. 8th International Conference on Theory and Applications of Satisfiability Testing (SAT-2005), pp. 173–186 (June 2005)
18. Oh, Y., Mneimneh, M.N., Andraus, Z.S., Sakallah, K.A., Markov, I.L.: AMUSE: a minimally-unsatisfiable subformula extractor. In: DAC '04. Proceedings of the 41st annual conference on Design automation, pp. 518–523. ACM Press, New York (2004)
19. Prosser, P.: Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence* 9, 268–299 (1993)
20. Torrens, M., Weigel, R., Faltings, B.: Java constraint library: Bringing constraints technology on the Internet using the java language. In: *Constraints and Agents: Papers from the 1997 AAAI Workshop*, Menlo Park, California, pp. 21–25 (1997)
21. Verfaillie, G., Lemaitre, M., Schiex, T.: Russian Doll Search. In: Proc. of the 13th National Conference on Artificial Intelligence, pp. 181–187 (1996)
22. Yokoo, M.: Asynchronous weak-commitment search for solving distributed constraint satisfaction problems. In: Montanari, U., Rossi, F. (eds.) CP 1995. LNCS, vol. 976, pp. 88–102. Springer, Heidelberg (1995)
23. Zhang, L., Malik, S.: Extracting small unsatisfiable cores from unsatisfiable Boolean formula. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, Springer, Heidelberg (2004)

Homogeneous Hierarchical Composition of Areas in Multi-robot Area Coverage

Sriram Raghavan¹ and Ravindran B²

¹ Intel India

sriram.raghavan@intel.com

² Department of Computer Science & Engineering, IIT Madras

ravi@cs.iitm.ernet.in

Abstract. Multi-robot area coverage poses several research challenges. The challenge of coordinating multiple robots' actions coupled with the challenge of minimizing the overlap in coverage across robots becomes even more complex and critical when large teams and large areas are involved. In fact, the efficiency critically hinges on the coordination algorithms used and the robot capabilities.

Multi-robot coverage of such large areas can be tackled by the divide-and-conquer policy; decomposing the coverage area into several small coverage grids. It is fairly simple to devise algorithms to minimize the overlap in small grids by making simple assumptions. If the overlap ratio of these small grids can be controlled, one may be able to integrate them appropriately to cover the large grid.

In this paper, we introduce homogeneous hierarchical composition grids to decompose a coverage area into several small coverage primitives with appropriately sized robot teams. These coverage grids are viewed as cells at a Meta level and composed hierarchically with such teams functioning as a single unit. We state and prove an associated theorem that provides very good scaling properties to large grids. We have performed simulated studies to validate the claims and study performance.

1 Introduction

Multi-Robot Area Coverage involves visiting every point within a given area by a team of mobile robots. Such tasks are typical to coordinated tasks such as Robotic vacuuming, Robotic de-mining or Robotic rescue. In such applications, it is sufficient if any one member of the team visits a particular point in the coverage area as repeated visits provides no additional information or value. Revisits are considered as overhead on the task completion. Such coverage tasks are usually characterized by few points for entry and no a priori knowledge about the terrain for the area. However, it is reasonable to assume that periphery can be identified through vision or radio beacons.

Consider a situation where a group of rescue robots form a rescue team to evacuate survivors from a building which is devastated by natural calamity. In such a situation, it is required that the team coordinates its actions so as to rescue as many survivors as

possible, quickly. We assume, for sake of simplicity, that all searches are restricted to searching on planar space. It is typical of such a scenario to round up the area (periphery of coverage area is known) and scan extensively within. In order to be effective, the team members must spread out after entering the area and avoid revisits to a location. Coordination also necessitates the need to set up a common reference during communication which requires a coordination architecture [1] to be built into these robot team members. This is then achieved thru exchange of messages to decide on future course of action. Since the robots are mobile, the coordination architecture must support one or more wireless communication technologies.

Once these are in place, the robots should use a common protocol or coordination algorithm to exchange their findings and status (robot states) to decide the optimal next step. While it is best if robots synchronized with each other after every step to take the optimal next step, it results in significant communication overhead. The tradeoff between the periodicity of communication and the level of optimality in robot actions is decided based on application needs. If the area is very vast, then the robots are divided into smaller teams and are made to cover smaller regions in parallel. This technique has three distinct advantages, viz., coordination in smaller teams is simpler and faster, the robots can cover the smaller regions using simple and efficient algorithms to minimize overlap and if regions do overlap, it is restricted to the smaller team and not the entire robot group.

Area coverage, in literature, is performed using two kinds of area decomposition: approximate cellular decomposition [4,10,14] where the coverage area is approximated as a grid of cells and exact cellular decomposition [2,7] where it is exactly mapped by one or more such grids. *The cell is the footprint of a robot and the area covered by it in one unit of time.* Increasing the size of the cell implies increase in robot coverage ability in unit time which may be required to support advanced applications.

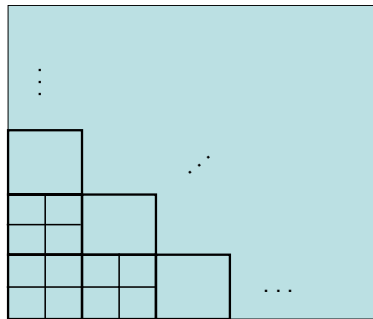


Fig. 1. Illustrating concept of hierarchy by building up using small areas

In this paper, we introduce the concept of teams of robots and coverage primitives which form the building blocks of coverage. These robots are simple in design with limited sensing and computational capabilities being easy to program and cheap to build. However, they cannot use conventional coordination techniques owing to their limited capabilities. This paper briefly discusses various coordination algorithms developed

by us to suit different contexts and robot capabilities. These algorithms quantitatively measure the overlap using a parameter called the `overlap_ratio` and attempt to minimize it using various strategies in the different contexts. A team will employ an appropriate algorithm to cover a primitive and move from one primitive to another after coverage. The algorithm is selected based on the application needs and individual robot capabilities. Since a team of robots move from one primitive to another during coverage, these primitives may be equated to ‘large’ cells being covered by robots with greater degrees of freedom. We use this concept and compose very large areas using primitives and divide the robots as appropriate directed by application needs. This will give rise to hierarchical composition of the area where we prove that the total `overlap_ratio` of the coverage area is the sum of `overlap_ratios` of immediate two sublevels levels and this result is scalable to any number of levels in hierarchy.

The rest of the paper is organized as follows. In section 2, we present the state-of-the-art in multi-robot area coverage. In section 3, we describe the different classes of multi-robot coordination algorithms we have developed that successively reduce the overlap in coverage across robots and also state the assumptions on which these algorithms are based. We also classify these algorithms and categorize the same. In section 4, we present the homogeneous hierarchical composition theorem and discuss the test cases studied. In section 5, we discuss the experimental setup, summarize the results and discuss our findings. In section 6, we conclude by providing a brief summary of the work achieved in this paper.

2 State-of-the-Art in Multi-robot Area Coverage

There are three dimensions in which mobile robot coordination for achieving a coverage task can be classified. They are: *Coverage Problem*, *Coordination Problem*, and *Communication Problem*. Cao et al. [13] provide a classification of multi-agent robotics along the dimensions of communication, computation and other capabilities. Choset surveys the area coverage problem [7] and introduces some *basic coverage heuristics*. Butler et al. [8] describe algorithms that guarantee coverage of rectilinear environments by a team of robots. Rekleitis et al [9] describe a graph-based multi-robot exploration and mapping approach which keeps two robots in closely-coupled coordination each robot is always in line-of-sight of the other.

Solanas and Garcia [2] present an unsupervised clustering algorithm that partitions the unknown space into as many cells as the number of mobile robots. The assignment of regions to the various robots is based on bids that are estimates of information gain traded-off against traveling costs to that region. Simmons et al. [11] describe a centralized exploration and mapping algorithm that uses maximum likelihood to find maps maximally consistent with the sensor data from the region. Zlot et al. [4] present a totally distributed exploration algorithm in mobile robots based on the market economy which minimizes traveling costs and maximizes information gain.

In ant-robot based terrain coverage [6], simple robots with minimal sensory capabilities perform at least once-coverage or continual coverage of an unknown terrain. The terrain is exactly decomposed into cells, each of which is the size of a robot. The work assumes that multiple robots may visit a single cell simultaneously without hindering the coverage path of other robots. Robots move in perfect synchronization

during coverage without communicating with one another but rely on pheromone trails left by other robots earlier at that location. The action selection mechanism is based on an arbitrary function used to select the action that minimizes some cost function known a priori. Kube and Bonabeau have also looked at ant-like movements for robot exploration [10] where a leader is elected and the remaining robots follow the leader along some arbitrary path. But the objective function in such techniques is either at least once coverage or continual coverage of a terrain, both of which are not the goal of our thesis. Our work attempts to avoid revisits to cells during terrain coverage whenever possible.

3 Multi-robot Area Coverage Algorithms

A coverage area can be visualized as a grid consisting of $M \times N$ cells, each of square geometry and identical size. Representing an area in this form is called the occupancy grid representation. In this representation, a zero denotes free unexplored space and a non-zero value denotes either a covered cell or obstacles, as the case may be. Typically, positive numbers are used to represent robot visits and negative numbers are used to represent obstacles. A team of M homogeneous mobile robots, which can communicate with each other, is deployed for coverage. Each robot is identified with a globally unique ID. When the robots communicate, they exchange ‘state’ information. We assume that the robots have the capability to sense the locations (or cell) and the boundaries when they reach them.

The presence of fewer entry points into the area requires that the team enter the region and spread out as quickly as possible to cover the grid while minimizing overlap. Overlap is defined as a visit by a robot to an already covered cell and by definition, it is cumulative. To measure this, we introduce a parameter called the `overlap_ratio` which is formally defined in eqn. (1). In order to minimize overlap, negotiation strategies must distribute work among the robots in a fair manner and ensure that robots independently cover as much area as possible before coordinating their actions to cover the remaining area. This requires robots to exchange and process a lot of information before they cover cell independently. It also necessitates robots to be predictable in their actions for which they must follow deterministic coverage patterns. As this is difficult to achieve and implement in simple communicating robots capable of scanning and maintaining cell status, we treat coordination as the exchange of a single message or a sequence of messages between the robots in the team, depending on their states.

$$\text{overlap_ratio} = \text{Number of cells in overlap} / \text{Total number of cells in grid} \quad (1)$$

Another challenge is that the robots often have knowledge only about the extent of this area and can sense/detect the adjacent cells. It forces the robots to take coverage decisions based on *their* knowledge of the environment. In such tasks, robots often do not have complete information about the area required to perform coverage optimally.

We have developed a class of multi-robot area coverage algorithms (on-line and off-line computations) to coordinate the robots for covering given areas minimizing repetitive visits to cells, measured using the `overlap_ratio`. Each algorithm is suited to specific types of robots and contexts. These are summarized in table 1.

Table 1. Area coverage algorithms and their requirements at a glance

Algorithm	Commu- nication	Collision Avoidance	Minimum Distance	Computa- tion
NCC	No	No	No	On-line
OSC	Yes	No	No	On-line
OSCARD	Yes	Yes	No	On-line
NJ	Yes	Yes	Yes	On-line
AAA	No	Yes	Yes	Off-line

In the NCC algorithm, each robot randomly decides its next step for movement. It is naïve, easy to implement and forms our basis for comparison with other multi-robot coverage algorithms. The next set of algorithms (except AAA) are all online algorithms based on the assumption that robots communicate at every step to synchronize their actions in order to minimize overlap. When a decision is taken, it is assessed by other team members thru inter-robot communication. If the action is acceptable to all robots in the team, then this action is performed by that robot. *Acceptable* refers to the act of ensuring that two or more robots are not present in the same cell at the same time. If such a situation occurs, a collision is deemed to have taken place.

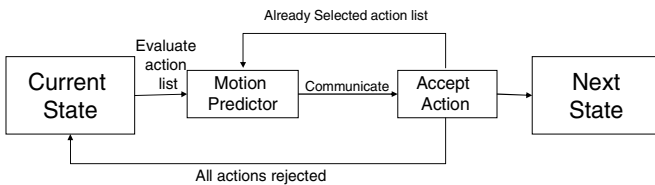


Fig. 2. Coordinated Robot Movement Strategy

The OSC algorithm employs one-step communication with other robots to avoid collisions. OSCARD employs one-step communication and assumes the capability for each robot to recognize already covered cells. Since, all robots are homogeneous, they cover a cell in identical manner and this information is exploited in this algorithm. The NJ algorithm maintains a fixed inter-robot distance in addition to having all capabilities and requirements of OSCARD. The threshold for inter-robot distance is determined by the grid size and number of robots in the team. Figure 2 summarizes the robot movement strategy of these online coordination algorithms.

Each algorithm is a different implementation of the “motion predictor module” which helps robots to avoid visiting already covered cells. Despite avoiding visited cells, a robot might get trapped and the algorithms have ways of breaking the dead-lock at the cost of increased overlap. The AAA is an offline algorithm that assumes global grid knowledge and computes each robot’s initial position in the grid using an equally-likely distribution. The robots then move in a deterministic manner to those

positions and perform coverage along the axis of the grid. Their orientation is known a priori and the robots maintain the same throughout coverage.

4 Homogeneous Hierarchical Composition of Areas

Typical area coverage scenarios require a team of robots to cover very large areas. In such scenarios minimizing global overlap with distributed coordination mechanisms can be very difficult. One solution to this problem would be to try and localize the coordination required by adopting a divide and conquer approach. We accomplish that by dividing our robots into teams and spread the teams out to cover smaller units of the grid in parallel. In the context of multi-robot area coverage, the coverage grid can be split into smaller grids of fixed sizes where overlap ratio is controlled and repeat that pattern until coverage is complete in the entire grid. Each team will have a leader who is responsible to coordinating his team's actions with other teams in order to minimize overlap. This leader can be elected using any known leader election [12,15] methods. Multiple teams could, therefore, work independently while coordinating among themselves while performing coverage. This has a significant impact on saving communication cost between the robots during coordination. In our work on area coverage, we have devised a methodology to scale the algorithms to cover very large grids using algorithms described in section 3.

In hierarchies, we define the smallest grid used to decompose the area as a *primitive* which is covered by a team of robots. A *team* is defined as a group of robots using the same algorithm covering the same grid. The coverage of each primitive is achieved by the team synchronously. Grids formed as a consequence of stacking the primitives together give rise to larger grids which are called *grid-cells* at the higher levels in the hierarchy. A grid-cell can also be considered as a primitive grid at these higher levels covered by a larger team of robots. Hence, the rules that apply to a primitive apply at all the intermediate levels in the hierarchy. This is illustrated in Fig. 3.

Several such teams of robots occupy and cover the primitives in an order directed by some meta-level area coverage algorithm and in effect cover the entire area. In the level immediately above the primitives in the hierarchy, each primitive can also be treated as a cell and the each team of robots covering the primitive may be *treated* as a single "more powerful" robot covering that "cell". We may then use the same set of algorithms to cover the area at the next higher level in hierarchy. The coverage algorithms used *within a team* and *across teams* are totally decoupled and the best combination may be employed to minimize overlap. It is then sufficient to compute the optimal decomposition of a given coverage grid in terms of these small grids and obtain their initial positioning. We now present the crux of our work and introduce the hierarchical composition theorem (H^2C theorem) for theoretically computing the overlap ratio in very large grids using the empirical results obtained from overlap ratio for small grids of different configurations.

Theorem 1. *The overlap_ratio in covering an $N \times N$ grid using some $n \times n$ as the coverage primitive for two levels in a hierarchical manner is given by sum of the overlap ratios in the grids at the two levels.*

Theorem 2. *The overlap ratio, as obtained from Homogeneous Hierarchical Composition Theorem, in covering a region using smaller coverage grids in a multi-level hierarchy is given by*

$$O(m): x_0 + x_1 + x_2 + \dots + x_m$$

where $O(m)$ is the overlap ratio obtained at the m^{th} level in the hierarchy and x_0 through x_m are the overlap ratios obtained at the corresponding levels using the primitive grids

Note: Formal proof to the theorems is given in Appendix 1.

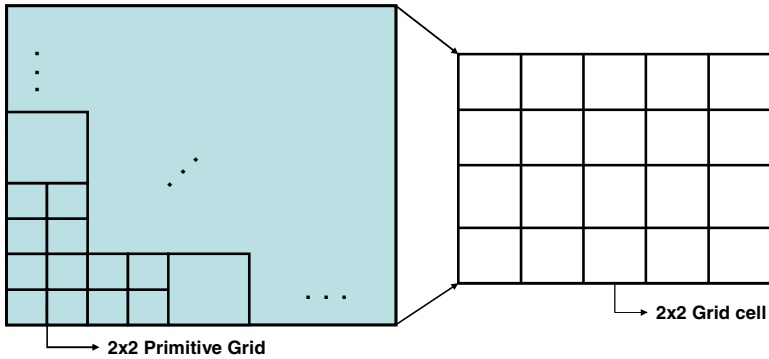


Fig. 3. Illustration of Hierarchical Area Composition and Meta-level view

4.1 Implication of Homogeneous Hierarchical Composition

An important consequence of the theorems stated above is that a grid may be decomposed into as many levels in hierarchy as required and in any order so as to minimize overlap. However, the number of levels in hierarchy will impact the number of robots needed and in turn the inter-robot communication cost to maintain synchrony. Another consequence is that, any coordination algorithm that satisfies the requirements in table 1 and performs cohesive team movement from one primitive to another can be used at any level(s) in coverage.

The theoretical and experimental results for `overlap_ratio` obtained through hierarchical decomposition deviate from one another as the experimental overlap had to take the team relocation into account while shifting from one sub-grid to another. These robot movements were assumed to be deterministic across the primitives and require communication between the various primitive team leaders to coordinate their respective team movements and minimize overlap. It can be shown that this deviation is bounded by a factor $m(n-1)/2n^2$, where m is size of small robot team and n is the size of primitive grid. For values of m and n chosen such that $m \leq n$, this factor is always less than 0.5. Therefore, the hierarchical composition framework effectively spreads the robots into teams across the grid and achieves coverage at acceptable levels of overlap ratio. Our framework was validated on large grids of size 1024×1024 for various team sizes and the results clearly indicated that the system can save over 90% of effort (computed as number of robot movements) even using the naïve NCC algorithm in covering the area.

4.2 Limitations of Homogeneous Hierarchical Composition

According to the theorems, we assume that the team of robots operating within one primitive grid function as a single unit at the immediate next higher level. This implies that while moving from one primitive to the next, all the robots must move as a cohesive unit from the current primitive to the same next primitive grid and in a deterministic manner. This requires the robots to maintain close coordination between them (even for naïve NCC coverage algorithm!). One should note that when the robot-collision avoidance is performed at an intermediate level in the hierarchy, it corresponds to a team of robots avoiding another team while shifting across primitives. This requires the leaders to communicate between themselves and maintain the required distance between their corresponding teams as dictated by the meta-level coverage algorithm. If the meta-level algorithm does not mandate communication, then leaders will perform basic communication to ensure that robot collisions and overlap across teams are minimized. While this operation amounts to overhead for using the hierarchical framework in area coverage, the alternative (direct coverage of the large grid) requires all robots to communicate with each other until they moved into mutually undisturbed positions. The overhead involved in achieving the latter far exceeds the cost of coverage in terms of number of steps to complete coverage or resources required. In comparison, the complexity would reduce by several orders of magnitude by using the leader election technique for inter-team coordination. It is always possible to restrict the number of such teams sent in to cover a given area and effectively reduce the communication overhead.

5 Homogeneous Hierarchical Composition Applied to large areas

The power of the H^2C theorem is highlighted when we study its performance in comparison to direct coverage of large grids using the same algorithms. In each of the following figures shown, the algorithm indicated was used in performing direct coverage as well as in hierarchical coverage. In the case of hierarchical coverage with several levels of hierarchy, the same algorithm was employed at all the levels with equal number of robots at the lower and each higher level.

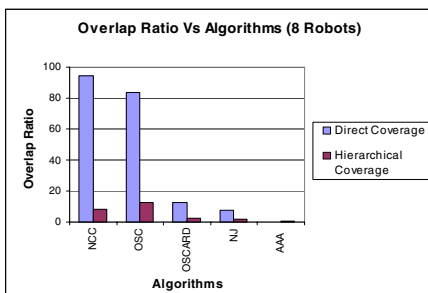


Fig. 4. Direct Vs Hierarchical Coverage of 64×64 Grid (8 Robots)

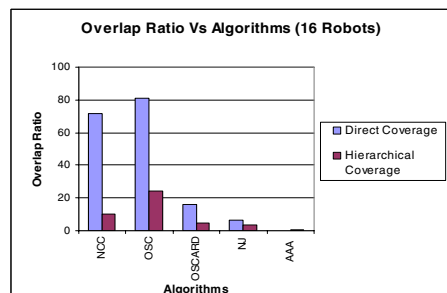


Fig. 5. Direct Vs Hierarchical Coverage of 64×64 Grid 16 Robots

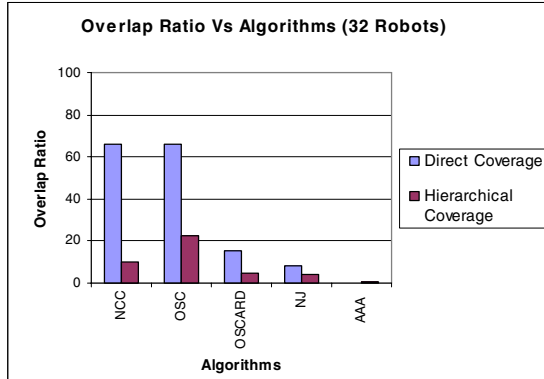


Fig. 6. Direct Vs Hierarchical Coverage of 64x64 Grid 32 Robots

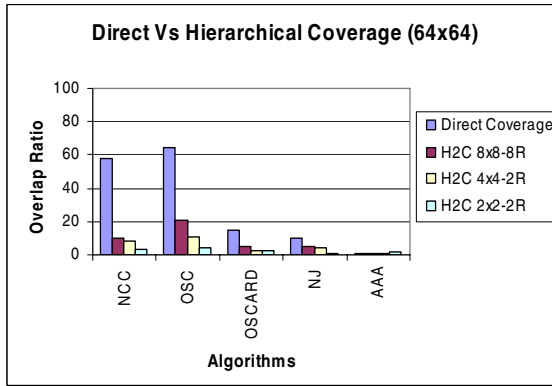


Fig. 7. Direct Vs Hierarchical Coverage of 64x64 Grid

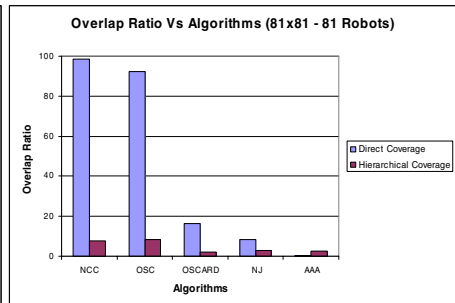
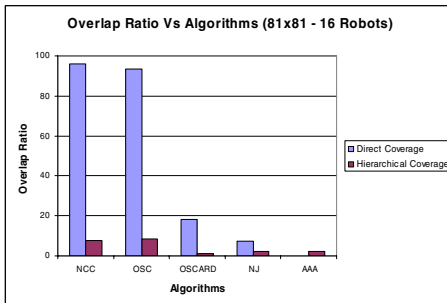


Fig. 8. Direct Vs Hierarchical Coverage of 81x81 Grid 16 Robots

Fig. 9. Direct Vs Hierarchical Coverage of 81x81 Grid Using 81 Robots

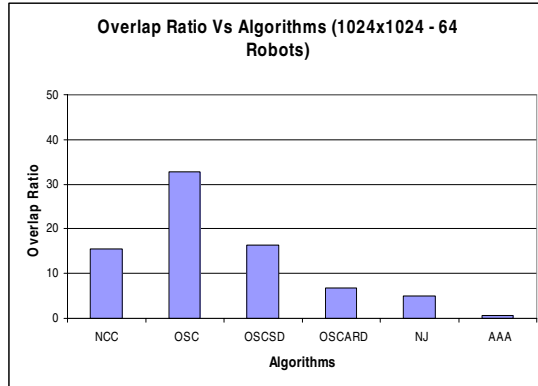


Fig. 10. Performance of Hierarchical Coverage of 1024×1024 Grid Using 64 Robots

5.1 Design of Experiments

It is noteworthy to mention that when the coverage related simulation experiments were conducted on grids of very large sizes (256×256 , 512×512 and 1024×1024 , for example), all algorithms other than the NCC algorithm did not run to completion even when simulated for over 36 hours. We inferred that this behavior was because all online algorithms barring the NCC algorithm require consensus through communication from all robots in the team at every step. This was significant communication overhead on the system and the robots were busy most of the time communicating to obtain acceptance. As a consequence, coverage rate was drastically slow and completion was never reached.

On the other hand, when the number of robots was decreased to an acceptable number, there are lesser number to perform coverage, most of which were attempting to cover the same section of the grid and unable to get out of this section. It must be

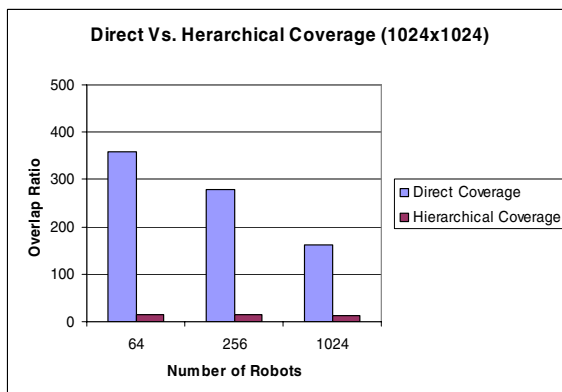


Fig. 11. Comparing Performance of Coverage for varying robot team sizes in 1024×1024 grid

noted that though the coverage algorithms (as compared with NCC) attempt to minimize visits to already covered cells, they still select randomly when there is more than one possibility. A comparison of the performance of direct coverage against hierarchical coverage is shown in Fig. 11 for 1024×1024 grid for varying number of robots using the NCC algorithm. As mentioned before, NCC algorithm was applied at all levels in the hierarchical case.

6 Summary and Conclusion

In this paper, we have discussed one approach to area coverage using multiple mobile robots. In our work, we developed a methodology to solve this problem in a coordinated manner using these robots. We briefly discussed the different coordination algorithms we designed with small robot teams and successively refined them to minimize overlap_ratio.

We then proposed a hierarchical framework for integrating these solutions for coverage of small areas to cover arbitrarily large areas and stated the Homogeneous Hierarchical Composition Theorem that states that the overlap ratio in coverage of a given area consisting of a grid of multiple primitive grids is the sum of the overlap ratios in coverage at the two levels. This result was further shown to be scalable to any number of levels in hierarchy.

The design of the experiments was explained and the results were presented. The performance of these algorithms was studied for varying number of robots in a team and for varying grid sizes and the various observations were listed. Results clearly indicate that this framework is very effective and for large grids, can save effort (measured by number of robot actions) even using simple coverage algorithms.

In future we plan to extend the results discussed in this paper to arbitrary sized areas and varying decompositions. Work is currently underway to extend the coordination algorithms to heterogeneous robots under lossy communication channels.

References

1. Raghavan, S., Ravindran, B.: Profiling Pseudonet Architecture for Coordinating Mobile Robots. In: Second International Conference on Communication System Software and Middleware (COMSWARE), Jan 7-12, Bangalore, India (2007)
2. Solanas, A., Garcia, M.A.: Coordinated multi-robot exploration through unsupervised clustering of unknown spaces. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), vol. 1, pp. 717–721 (2004)
3. Farinelli, A., Iocchi, L., Nardi, D.: Multirobot Systems: A Classification Focused on Coordination. *IEEE Transactions on Systems, Man, Cybernetics-Part B* 34(5) (2004)
4. Zlot, R., Stentz, A., Dias, M., Thayer, S.: Multi-Robot Exploration Controlled By A Market Economy. In: Proceedings of IEEE International Conference on Robotics and Automation (ICRA) (2002)
5. Dudek, G., Jenkin, M., Milios, E.E.: Robot Teams: From Diversity to polymorphism - A Taxonomy of Multirobot Systems, AK Peters, Wellesley MA (2002)

6. Koenig, S., Liu, Y.: Terrain Coverage with Ant Robots: A Simulation Study. In: Proceedings of the International Conference on Autonomous Agents (AGENTS 2001), pp. 600–607 (2001)
7. Choset, H.: Coverage for Robotics - A Survey of Recent Results. In: Annals of Mathematics and Artificial Intelligence. vol. 31, pp. 113–126. Kluwer, Norwell, MA (2001)
8. Butler, Z.J., Rizzi, A., Hollis, R.L.: Complete Distributed Coverage in Rectilinear Environments. In: Peters, A.K.(ed.) Proc. of the Workshop on the Algorithmic Foundations of Robotics (January 2001)
9. Rekleitis, I., Dudek, G., Milios, E.E.: Graph-based exploration using multiple robots. In: Proceedings of the Fifth International Symposium of Distributed Autonomous Robotic Systems (DARS), October 4-6, 2000, Knoxville, Tennessee, pp. 241–250 (2000)
10. Kube, C.R., Bonabeau, E.: Cooperative Transport by ants and robots. Journal of Robotics and Autonomous Systems 30(1), 85–101 (2000)
11. Simmons, R., Burgard, W., Moors, M., Fox, D., Thrun, S.: Collaborative Multi-Robot Exploration. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), San Francisco, CA (2000)
12. Stoller, S.D.: Leader Election in Asynchronous Distributed Systems. In: Proceedings of IEEE Transactions on Computers, vol. 49(3), pp. 283–284 (March 2000)
13. Cao, Y.U., Fukanga, A., Kahng, A.: Cooperative Mobile Robotics: Antecedents and Directions. Autonomous Robotics 4, 1–23 (1997)
14. Rekleitis, I.M., Dudek, G., Milios E.E.: Multi-robot exploration of an unknown environment: Efficiently Reducing the Odometry Error. In: Proceedings of 15th International Joint Conference on Artificial Intelligence (IJCAI-97), pp. 1340–1345, Nagoya, Japan (August 1997)
15. Francis, P., Saxena, S.: Optimal Distributed Leader Election Algorithm for Synchronous Complete Network. IEEE Transactions on Parallel and Distributed Systems 7(3) (1996)

Appendix 1: Homogeneous Hierarchical Composition Theorem

Proof for Theorem 1: Consider a square grid of area $N \times N$ being covered by a team of M robots; let us assume that the grid can be composed using primitives of area $n \times n$. Then the $N \times N$ grid consists of $k^2 = (N \times N)/(n \times n)$ primitives (say). We position these k^2 grids in the form of a square to cover the entire grid. Let $M = a \times b$ and let each $n \times n$ grid be covered by ‘a’ robots. Then there are ‘b’ such teams to cover the $N \times N$ grid. Let these b teams be deployed in the k^2 grid for coverage. The problem is illustrated in Fig 12.

Note: Although the proof refers to square grids for ease of exposition, these results apply equally well to rectangular shaped grids.

Let us suppose that the coverage of k^2 grid using a team of ‘b’ robots results in an overlap ratio of x_1 . We obtain an overlap ratio of x_0 in the coverage of each primitive grid by a team of ‘a’ robots. Coverage of $k \times k$ grid necessitates the coverage of each primitive $n \times n$ and together, they guarantee the coverage of the $N \times N$ grid, as specified. Hence, the total number of cells in overlap is given by –

$$\begin{array}{l} \text{Number of cells in overlap} \\ \text{for the primitive grid coverage} \end{array} = (x_0 \times n^2) \times k^2$$

Number of grid-cells in overlap during coverage of $k \times k$ grid = $(x_1 \times k^2)$

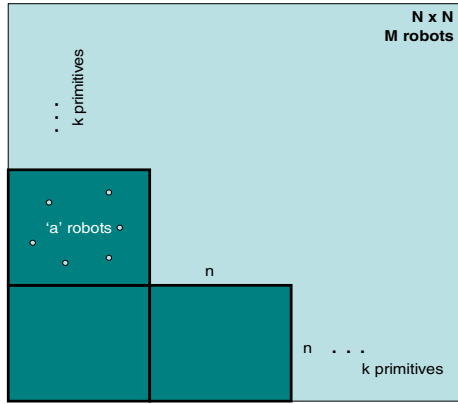


Fig. 12. Illustration of the H²C theorem

Each cell of $k \times k$ grid is an $n \times n$ grid. Hence number of cells in overlap is given by – = $(x_1 \times k^2) \times n^2$

Total number of cells in overlap = $(x_0 + x_1) \times k^2 \times n^2$

Overlap ratio (By Definition) = $(\text{Total number of cells in overlap} / \text{Total number of cells})$
 = $(x_0 + x_1) \times k^2 \times n^2 / N^2$
 = $(x_0 + x_1) \times N^2 / N^2$
 = $(x_0 + x_1)$

This concludes the proof for Homogeneous Hierarchical Composition Theorem for obtaining the overlap in a large grid hierarchical composition of primitive grids. □

Proof for Theorem 2: Proof by Principle of Mathematical Induction (PMI)

P(1): The theorem holds true for single level in hierarchy.

O(1) = $x_0 + x_1$

Proof: Proved in Theorem 1. Therefore P(1) is true.

At the induction step, assume that the statement is true for some natural number m.

P(m): Theorem holds true for m levels in the coverage hierarchy. Therefore O(m): $x_0 + x_1 + x_2 + \dots + x_m$ is true.

To show that P(m+1) is true whenever P(m) is true. Then by PMI, statement P(N) is true for all natural numbers N.

P(m+1): The theorem holds for m+1 levels in coverage hierarchy whenever P(m) is true.

$$O(m+1): x_0 + x_1 + x_2 + \dots + x_m + x_{(m+1)}$$

From P(m), it is evident that any coverage grid can be composed for m levels in hierarchy and their overlap ratio can be obtained as the sum of overlap ratio at the corresponding levels. Let us now construct a grid of r^2 cells, each of which is an ‘m’ level coverage grid in hierarchy with a total of L cells. Let the actual number of cells in its side be M. Then we have the relation,

$$M^2 = r^2 \times L^2$$

Overlap ratio obtained in the L x L grid is given by –

$$O_L = [x_0 + x_1 + x_2 + \dots + x_m]$$

Let the overlap at the highest level in hierarchy (m+1) be $x_{(m+1)}$. The total number of cells in overlap is then given by,

$$\begin{aligned} \text{Overall Overlap} &= [(x_0 + x_1 + x_2 + \dots + x_m) \times L^2] \times r^2 + [x_{(m+1)} \times r^2] \times L^2 \\ &= [x_0 + x_1 + x_2 + \dots + x_m + x_{(m+1)}] \times r^2 \times L^2 \\ &= [(x_0 + x_1 + x_2 + \dots + x_m + x_{(m+1)}) \times r^2 \times L^2 / M^2] \end{aligned}$$

Substituting from equation (1), we obtain,

$$\begin{aligned} \text{Overlap ratio} &= [x_0 + x_1 + x_2 + \dots + x_m + x_{(m+1)}], \\ &\text{which proves our P(m+1) statement.} \end{aligned}$$

Therefore P(m+1) is true whenever P(m) is true. Hence, by PMI, statement P(N) is true for all Natural Numbers. This concludes the proof of Generalized H²C theorem. □

Formalizing the Abstraction Process in Model-Based Diagnosis

Lorenza Saitta¹, Pietro Torasso², and Gianluca Torta²

¹ Dipartimento di Informatica, Università del Piemonte Orientale,
Via Bellini 25/G, 15100 Alessandria, Italy
saitta@mf.n.unipmn.it

² Dipartimento di Informatica, Università di Torino
Corso Svizzera 185, 10149 Torino, Italy
{torasso,torta}@di.unito.it

Abstract. Several theories have been proposed to capture the essence of abstraction. Among these, the \mathcal{KRA} model offers a framework where a set of generic abstraction operators allows abstraction to be automated.

In this paper we show how to describe component-based abstraction for the Model-Based Diagnosis task within the \mathcal{KRA} framework, and we discuss the benefits of such a formalization.

The clear and explicit partition of the system model into different levels required by \mathcal{KRA} (going from the perception level up to the theory level) opens the way to explore richer and better founded kinds of abstraction to apply to the MBD task.

Another noticeable advantage is that, by suitably personalizing the generic abstraction operators of \mathcal{KRA} , the whole abstraction process, from the definition of abstract (macro)components to the computation of their behaviors starting from those of the ground components, can be performed automatically in such a way that important relationships between ground and abstract diagnoses are guaranteed.

1 Introduction

Abstraction, intended as the ability to forget irrelevant details and to find simpler descriptions, is a pervasive activity in human perception and reasoning. In the Artificial Intelligence community, abstraction has been investigated mostly in problem solving [1], planning [2], Model-Based Diagnosis [3] and in problem reformulation [4]. Various abstraction theories have been proposed in the attempt to capture a general notion of abstraction. These theories are either syntactic [5], or semantic [6], but they fail to characterize the practical aspects of the abstraction process. Following an idea of Korf [7], Saitta and Zucker [8] have proposed a model of representation change that includes both syntactic reformulation (different formalisms, same information content) and abstraction (same formalism, but reduced information content). The model, called \mathcal{KRA} (from Knowledge Reformulation and Abstraction), is designed to help both the conceptualization phase of a problem and the automatic application of abstraction operators.

In this paper we use the \mathcal{KRA} model to formalize abstraction in Model-Based Diagnosis (MBD). Starting from the seminal work by Mozetic [3], the main motivation for deriving abstract models for MBD is to reduce the computational complexity of the diagnostic process without losing (too much) diagnosability.

We expect that formalizing the abstraction process in MBD within the \mathcal{KRA} framework will open the way to explore richer and better founded kinds of abstraction to apply to the MBD task. In particular, \mathcal{KRA} requires that the system model is explicitly described at different levels, ranging from the perception level (that is peculiar to \mathcal{KRA}) up to the semantic and syntactic levels; this should make possible to express the given/desired abstractions at the appropriate level(s) and to automatically propagate their effects as induced abstractions on the other levels while preserving some desired properties.

To give a first demonstration of these benefits, in this paper we focus on a well-known kind of abstraction that consists in aggregating elementary system components into an *abstract* component, with the effect of merging combinations of behaviors of the original components into a single abstract behavior of the abstract component. Component aggregation will be modeled, inside the \mathcal{KRA} framework, through a set of *aggregation* operators, which automatically perform all the required transformations, once the components to be aggregated are specified. These transformations change the original *ground* description of the system to be diagnosed into an *abstract* one involving macrocomponents.

In most applications of abstraction to MBD (e.g. [9, [10], [11]), the authors aim to generating abstract models where the discrimination power among alternative diagnoses is fully preserved. We will briefly discuss how this particular property can be easily guaranteed for aggregation of components within our approach.

2 A Running Example

In the body of the paper we will use a running example to clarify the notions we will introduce. In particular, we will consider the hydraulic system reported in Figure 1, which is the same used in [10]. It includes five types of components, i.e., *pipes*, *valves*, *joins*, *splits*, and *pumps*. Each component has *ports* that connect components with each other (internal ports), or connect components to the environment (external ports). In the case of pipes we have for each pipe two inputs and two outputs, where *flow* and *pressure* (resistance) can be measured respectively. Components can have *commands* applied to them, whose action is set from the outside; in particular, each valve has a command intended to set the valve to open or closed.

Each component can exhibit one among a set of pre-defined behaviors, each one described by a model expressed in terms of qualitative deviations [12].

In Figure 3 the possible behaviors of pipes, valves, pumps, splits and joins are reported. For example, when the pipe is in the behavioral mode *PC* (partially clogged), the qualitative deviation Δf_{out} of the flow at the end of the pipe is the same as the qualitative deviation Δf_{in} of the flow at the beginning of the pipe.

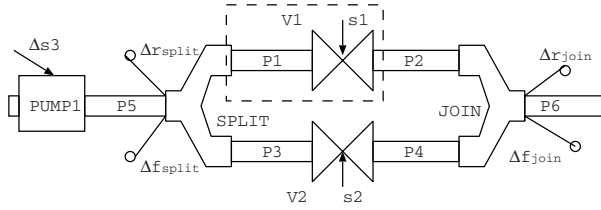


Fig. 1. Hydraulic system used as a running example

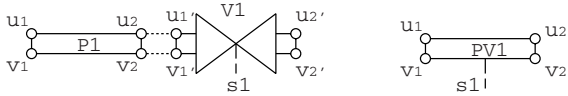


Fig. 2. Fragment of the system used as an example

For the pressure, instead, we have that $\Delta r_{in} = \Delta r_{out} \oplus +$ where \oplus is the addition in the *sign algebra* (see again [12]). In other words, the pressure is qualitatively increased at one end of the pipe with respect to the other end.

In the rest of the paper we will assume that a pipe has one nominal behavior (*OK*) and three faulty modes *PC* (partially clogged), *CL* (clogged) and *BR* (broken), while the valve can be in the *OK* mode (and, in this case, it behaves in a different way depending on the external command *s* set to *open* or *closed*), in the *SO* (stuck open) mode, or in the *SC* (stuck closed) mode. Splits and joins are supposed to always be in the *OK* state, whereas the pumps, apart from the nominal behavioral mode, may be in one of the following faulty modes: blocked (*BL*), underpumping (*UP*) and overpumping (*OP*).

In the system considered in this paper, let us assume that the only observable parameters are Δf_{split} , Δr_{split} , Δf_{join} and Δr_{join} , i.e., the qualitative deviations of flow and resistance (pressure) before the split and after the join components. We also assume that the s_1 command is set to *open* whereas s_2 is set to *close*.

In Figure 2 an expanded view of the subsystem enclosed in the dotted rectangle in Figure 1 is reported. From now on, we will often refer to it for providing detailed descriptions. In the left part we see that components *P1* and *V1* are connected together by means of ports (in particular, ports u_2 and v_2 of *P1* are attached to ports u'_1 and v'_1 of *V1* respectively). Since our goal in the abstraction process is to aggregate elementary components into abstract components, we aim at automatically synthesizing the behaviour of such abstract components. In particular we could be interested in abstracting the hydraulic system model by aggregating the components *P1* and *V1* into the abstract component *PV1* reported in the right portion of figure 2. In such an abstract component we still have ports u_1 , v_1 , u_2 and v_2 that allow the abstract component to be connected with the rest of the system, while u_2 , v_2 , u'_1 and v'_1 have been hidden

$OK(Pipe)$	$\Delta f_{out} = \Delta f_{in} ; \Delta r_{in} = \Delta r_{out}$
$PC(Pipe)$	$\Delta f_{out} = \Delta f_{in} ; \Delta r_{in} = \Delta r_{out} \oplus +$
$BR(Pipe)$	$\Delta f_{out} = - ; \Delta r_{in} = -$
$CL(Pipe)$	$\Delta f_{out} = \Delta f_{in} = - ; \Delta r_{in} = +$
$OK(Valve)$	$s(open) \Rightarrow \Delta f_{out} = \Delta f_{in} ; \Delta r_{in} = \Delta r_{out}$ $s(close) \Rightarrow \Delta f_{out} = \Delta f_{in} = - ; \Delta r_{in} = +$
$SO(Valve)$	$\Delta f_{out} = \Delta f_{in} ; \Delta r_{in} = \Delta r_{out}$
$SC(Valve)$	$\Delta f_{out} = \Delta f_{in} = - ; \Delta r_{in} = +$
$OK(Pump)$	$\Delta f_{out} = \Delta s \ominus \Delta r \oplus \Delta f_{in}$
$UP(Pump)$	$\Delta f_{out} = \Delta s \ominus \Delta r \oplus \Delta f_{in} \oplus -$
$OP(Pump)$	$\Delta f_{out} = \Delta s \ominus \Delta r \oplus \Delta f_{in} \oplus +$
$BL(Pump)$	$\Delta f_{out} = -$
$OK(Join)$	$\Delta f_{out} = \Delta f_{in1} \oplus \Delta f_{in2} ; \Delta r_{out} = \Delta r_{in1} = \Delta r_{in2}$
$OK(Split)$	$\Delta f_{in} = \Delta f_{out1} \oplus \Delta f_{out2} ; \Delta r_{in} = \Delta r_{out1} = \Delta r_{out2}$

Fig. 3. Example of behavioral models of components. The term Δr for the pump has to be intended as the resistance on the output, whereas the term Δs is the command applied to the pump.

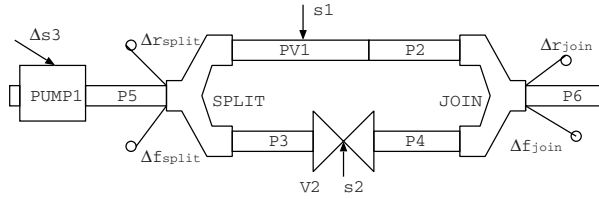


Fig. 4. Result of the aggregation of $P1$ and $V1$ into $PV1$

in the abstraction process. The abstract component $PV1$ has an external command s_1 that is exactly the same s_1 controlling the valve $V1$. The result of this abstraction step is reported in Figure 4.

In the rest of the paper we will show how all the needed transformations for synthesizing the abstract component can be done automatically by exploiting appropriate *aggregation* operators defined in the \mathcal{KRA} framework. One of the most critical points is the (automatic) synthesis of the abstract behaviors of the abstract component by merging combinations of behaviors of the original components, which will be addressed in Section 6.

It is worth noting that once we have an abstract model of the hydraulic system where $P1$ and $V1$ are replaced by the abstract components $PV1$, further abstractions are possible, for example by aggregating the abstract component $PV1$ with $P2$ into a new more abstract component $PVP1$. In section 7 we will show how far we can go with the abstraction process in the model of the hydraulic system and we will also provide information about the behaviors of the abstracted components.

3 The \mathcal{KRA} Model of Abstraction

In this section we briefly describe the \mathcal{KRA} model of abstraction. More details can be found in [13]. In the literature, transformations toward a more abstract

world have been defined between theories [14], languages [5] or databases [6]. Defining abstraction at the theory level (formulas are abstracted into formulas) is too powerful and hence of difficult application. More acceptable is the definition at the language level (predicate to predicate) and at the database level (table to table). However, all these proposals, trying to capture a very general notion of abstraction, may become underconstrained and may suffer from the problem of generating inconsistent abstract descriptions [15].

In order to face this problem, the \mathcal{KRA} model proposes to ground abstraction on perception (information coming from the outside world). Perception is a part of a more comprehensive representation framework $\mathcal{R} = (\mathcal{P}, \mathcal{D}, \mathcal{L}, \mathcal{T})$, describing a world \mathcal{W} at four levels: perception (\mathcal{P}), database (\mathcal{D}), language (\mathcal{L}) and theory (\mathcal{T}). Informally, the perception acts as the interface between the system under consideration and the outside world, and consists of two parts: the first contains the ontology of the domain (generic *types* of objects, their attributes and their relations), whereas the second one contains the description of the specific system to be analysed, including the actual objects, the actual system topology, the command values, the measurements provided by the sensors, and the tuples of objects satisfying the functions and relations definitions. This second part is grouped under the name *observations*.

More precisely, a perception $\mathcal{P} = (\mathbf{OBJ}, \mathbf{ATT}, \mathbf{FUNC}, \mathbf{REL}, \mathbf{OBS})$ is a 5-ple, where **OBJ** contains the types of objects considered in \mathcal{W} , **ATT** denotes the types of attributes of the objects, **FUNC** specifies a set of functions (in particular the *sensors* used to acquire information from \mathcal{W} and the *control* devices), and **REL** is a set of relations among object types. These sets can be expressed as follows:

- **OBJ** = $\{\mathbf{TYPE}_i | 1 \leq i \leq N\}$
- **ATT** = $\{A_j : \mathbf{TYPE}_j \rightarrow A_j | 1 \leq j \leq M\}$
- **FUNC** = $\{f_k : \mathbf{TYPE}_{i_k} \times \mathbf{TYPE}_{j_k} \times \dots \rightarrow C_k | 1 \leq k \leq S\}$
- **REL** = $\{r_h \subseteq \mathbf{TYPE}_{i_h} \times \mathbf{TYPE}_{j_h} | 1 \leq h \leq R\}$

The set **OBS** cannot be defined in general, because it refers to a particular system. An example of it will be provided in Section 5.

Going back to the representation framework \mathcal{R} , the database \mathcal{D} contains the extensional description of the system and of the observations, as well as any intermediate and final results generated by automated reasoning processes that may take place at the language and theory levels. The language \mathcal{L} is used to intensionally describe the contents of \mathcal{D} and for performing inferences. Finally, the theory \mathcal{T} contains both general and domain-specific knowledge, independent from the specific system under analysis. The content of the theory is described via the language \mathcal{L} .

Let us consider now two representation frameworks, $\mathcal{R}_g = (\mathcal{P}_g, \mathcal{D}_g, \mathcal{L}_g, \mathcal{T}_g)$ and $\mathcal{R}_a = (\mathcal{P}_a, \mathcal{D}_a, \mathcal{L}_a, \mathcal{T}_a)$. We call \mathcal{R}_g a *ground* framework. An abstraction mapping can be defined between \mathcal{R}_g and \mathcal{R}_a . Once \mathcal{P}_g is given, any particular system can be obtained by assigning values to an appropriate set of variables (ranging over objects, attributes, pairs in functions and tuples in relations). Each assignment to all of these variables is a *configuration* γ_g . Let Γ_g be the set of

all the possible ground configurations. In an analogous way, we can define the set Γ_a of all possible abstract configurations (for more details, see [13]). In other words, a configuration is any of the possible worlds perceivable with \mathcal{P}_g , including measurements. The set **OBS** is exactly one among the possible configurations, i.e., the one that is actually perceived. Using the above notions, we define an abstraction as follows:

Definition 1. *Given two representation frameworks \mathcal{R}_g and \mathcal{R}_a , if there exists a mapping $\Gamma_g \rightarrow \Gamma_a$, such that the mapping associates subsets of Γ_g to single elements of Γ_a , we say that \mathcal{R}_a is more abstract than \mathcal{R}_g .*

In Definition 1 abstraction is seen as a *relative* rather than an absolute notion, and is defined between representation frameworks and not single objects. The analysis of the meaning and implications of this definition is out of the scope of this paper, and we will rely on its intuitive understanding. Substantially, the definition says that abstraction performs some kind of information aggregation, simplifying thus representation, and, possibly, the reasoning about configurations (we refer again the reader to [13] for more details).

In practice, we only consider mappings that can be obtained from the application of a set of *abstraction operators*, which generate abstract configurations starting from ground ones. Some operators can be domain-independent, and some are domain-specific. Considering abstraction as a process consisting of operators application has the limit that not all possible mappings between \mathcal{R}_g and \mathcal{R}_a can be defined, but it has the advantage that general properties of abstractions (e.g. *compatibility*, see [13]) as well as task-specific properties of abstractions (e.g. *diagnosability*) can be enforced simply by putting constraints on operators. Finally, abstraction is a *transitive* relationship, and it induces a *partial order* among representation frameworks.

The introduction of operators provides a constructive semantics to abstraction. In fact, constraining the abstract framework to be *constructed* through operators has both the conceptual advantage of recognizing that abstraction is the result of a *process* rather than being a *state*, and the practical advantage to make abstraction operational.

4 Model-Based Diagnosis

In this section some basic notions of the Model-Based Diagnosis approach are provided for the sake of self-consistency. As typical in MBD, a system model is defined in terms of the system components and their connections.

Definition 2. *A System Description (SD) is a triple (COMPS, TOP, DT) where:*

- COMPS is the set of components. For each component c a set of predicates $BM_i(c)$ can be used to state that c is in behavioral mode BM_i ($1 \leq i \leq m_c$), where m_c is the number of behaviors of component c (there must be at least

a predicate *OK* for denoting the nominal behavior and possibly one or more predicates denoting faulty behaviors). The ports and exogenous commands of each component are also defined, through suitable ground atoms.

- *TOP* is a set of ground atoms defining the connections between components ports (i.e. the topology of the system). The topology partitions the components ports into P_{int} (internal ports that connect components with each other) and P_{ext} (external ports that connect components with the external environment)
- *DT* (Domain Theory) is a set of logical formulas representing the relation between the behavioral modes of components and the values of their ports and exogenous commands.

The above characterization of system description is able to capture a wide variety of static models, and in particular it does not require that the model is deterministic.

A status S of the system is a set of ground atoms s.t. for each $c \in COMPS$, S contains exactly one element of the form $BM_i(c)$.

From the definition above it is easy to derive the more general notion of subsystem, intended as a subset of components with their connections. In particular, a subsystem Σ involving subset $COMPS_\Sigma \subseteq COMPS$ will have an associated internal topology TOP_Σ , a Domain Theory DT_Σ , a set of ports and a set of exogenous commands. In particular, the set of external ports $P_{\Sigma,ext}$ contains all the ports of $COMPS_\Sigma$'s components that are connected with the external environment or with components that don't belong to Σ , while the set of internal ports $P_{\Sigma,int}$ contains the other ports of $COMPS_\Sigma$ components.

A status S_Σ of subsystem Σ is a set of ground atoms s.t., for each $c \in COMPS_\Sigma$, S_Σ contains exactly one element of the form $BM_i(c)$.

We are now ready to formalize the notion of *Diagnostic Problem* and of *Diagnosis*.

Definition 3. A Diagnostic Problem is a triple $DP = (SD, \mathcal{X}, \mathcal{O})$ where:

- SD is a System Description
- \mathcal{X} is a set of ground atoms denoting the values of all the external commands
- \mathcal{O} is a set of ground atoms expressing measurements on a subset PO of the components ports

Definition 4. Given a diagnostic problem DP , a (consistency-based) diagnosis for DP is a status D of the system s.t.:

$$DT \cup \mathcal{X} \cup \mathcal{O} \cup D \not\vdash \perp$$

The above definition corresponds to the classical characterization of consistency based diagnosis, requiring that the assignment of a behavioral mode to each component $c \in COMPS$ is logically consistent with \mathcal{X} and \mathcal{O} under constraints imposed by DT .

The observability degree of the system is determined by the subset PO of components ports that are actually observed. It is worth noting that, even though perfect observability does not guarantee that a unique diagnosis can be found, in general the lower is the observability, the larger the number of diagnoses.

The following definition introduces a notion of indiscriminability among states of a subsystem where the external ports of the subsystem are considered to be observable while the measurements on the internal ports of the subsystem, if any, are ignored.

Definition 5. Let Σ be a subsystem and \mathcal{X}_Σ be a set of ground atoms denoting the values of the external commands of Σ . We say that two states S_Σ, S'_Σ of Σ are \mathcal{X}_Σ -indiscriminable iff the following holds:

$$DT_\Sigma \cup \mathcal{X}_\Sigma \cup S_\Sigma \cup \mathcal{P}_\Sigma \vdash \perp \Leftrightarrow DT_\Sigma \cup \mathcal{X}_\Sigma \cup S'_\Sigma \cup \mathcal{P}_\Sigma \vdash \perp$$

where \mathcal{P}_Σ is any set of ground atoms expressing measurements on the external ports $P_{\Sigma,ext}$ of Σ .

According to this definition, two states S_Σ, S'_Σ of Σ are indiscriminable iff given any set of values measured on the external ports of the subsystem, S_Σ and S'_Σ are either both consistent or inconsistent with such measurements (under the constraints imposed by DT_Σ and \mathcal{X}_Σ).

5 The \mathcal{KRA} Model Applied to MBD

In this section we will show how the \mathcal{KRA} model can be applied to MBD, by using the running example introduced before. In particular, in this section we show how to capture the different entities needed for modeling the domain at the various representation levels. In this section we will refer, for the sake of exemplification, to the fragment of hydraulic system described in Figure 2, namely a pipe connected to a valve.

Let $\mathcal{R}_g = (\mathcal{P}_g, \mathcal{D}_g, \mathcal{L}_g, \mathcal{T}_g)$ be the ground representation framework. The ground perception $\mathcal{P}_g = (\mathbf{OBJ}_g, \mathbf{ATT}_g, \mathbf{FUNC}_g, \mathbf{REL}_g, \mathbf{OBS}_g)$ can be specified as follows:

- $\mathbf{OBJ}_g = \mathbf{COMP}_g \cup \mathbf{CTRDEV} \cup \mathbf{PORT} \cup \dots$ where $\mathbf{COMP}_g = \mathbf{PIPE} \cup \mathbf{VALVE}$
- $\mathbf{ATT}_g = \{ \text{ObjType}_g : \mathbf{OBJ}_g \rightarrow \{comp, pipe, valve, ctrdev, port, \dots\},$
 $\text{Direction} : \mathbf{PORT} \rightarrow \{in, out\},$
 $\text{Observable} : \mathbf{PORT} \rightarrow \{yes, no\},$
 $\text{MeasureType} : \mathbf{PORT} \rightarrow \{pressure, flow\} \}$
- $\mathbf{FUNC}_g = \{ Bp : \mathbf{PIPE} \rightarrow \{ok, pc, br, cl\},$
 $Bv : \mathbf{VALVE} \rightarrow \{ok, so, sc\},$
 $\text{Command} : \mathbf{CTRDEV} \rightarrow \{open, close\},$
 $\Delta FlowValue : \mathbf{PORT} \rightarrow \{+, 0, -\},$
 $\Delta PressureValue : \mathbf{PORT} \rightarrow \{+, 0, -\} \}$
- $\mathbf{REL}_g = \{ \text{port-of} \subseteq \mathbf{PORT} \times \mathbf{COMP}_g,$
 $\text{connected} \subseteq \mathbf{PORT} \times \mathbf{PORT},$
 $\text{controls} \subseteq \mathbf{CTRDEV} \times \mathbf{COMP}_g \}$

The set of objects is the set of possible system components, which includes pipes (**PIPE**), valves (**VALVE**), control devices (**CTRDEV**) and ports (**PORT**)¹. The set \mathbf{ATT}_g includes the type of the objects ObjType_g , the

¹ \mathbf{COMP}_g includes also pumps (**PUMP**), splits (**SPLIT**) and joins (**JOIN**) if the whole hydraulic system is considered.

Direction of ports, whether a port is *Observable*, and the type of sensors possibly attached to the port, namely *MeasureType*. For each attribute A , Λ_A denotes its range/set of values.

FUNC_g includes the behavioral modes of pipes (*Bp*) and valves (*Bv*), the function that opens/closes the valves (*Command*) and the qualitative measurements on ports (*ΔFlowValue* and *ΔPressureValue*).

Finally, **REL_g** includes three relations, namely, **port-of** (a port is attached to a component), **connected** (two ports of different components are connected), and **controls** (a device control is associated to a component).

The set **OBS_g** contains the actual objects and their topology, and the measurements provided by the sensors. Considering the fragment in Figure 2, we obtain: **OBS_g** = $\{(P1, V1, s_1, u_1, v_1, u_2, \dots), (ObjType(P1) = pipe, ObjType(u_1) = port, \dots), (Direction(u_1) = in, Direction(v_1) = in, \dots), (Observable(u_1) = yes, \dots), (MeasureType(u_1) = flow, \dots), Command(s_1) = open, (\Delta FlowValue(u_1) = +, \Delta PressureValue(v_1) = +, \dots), (\mathbf{port-of}(u_1, P1), \dots), (\mathbf{connected}(u_2, u'_1), \dots), controls(s_1, V1)\}$ All values provided by the perception are memorized in the database \mathcal{D}_g , which contains the tables:

- **TableObj** = (*obj, objtype, direct, obser, measuretype*), which describes objects of the actual system and their attributes,
- **TablePortOf** = (*port, comp*), which states to what component a given port is attached to,
- **TableConnected** = (*port, port*), which describes what ports are connected with each other, and
- **TableControls** = (*device, comp*), which describes on what component a control devices acts upon.

In the table **TableObj** some of the entries can be set to *N/A* (not applicable), as not all attributes are meaningful for all objects. For the sake of space, only a few rows of the table **TableObj** and the table **TableConnected** are provided as exemplification.

TableObj

<i>obj</i>	<i>objtype</i>	<i>direct</i>	<i>obser</i>	<i>measuretype</i>
<i>P1</i>	<i>pipe</i>	<i>N/A</i>	<i>N/A</i>	<i>N/A</i>
<i>V1</i>	<i>valve</i>	<i>N/A</i>	<i>N/A</i>	<i>N/A</i>
<i>u₁</i>	<i>port</i>	<i>in</i>	<i>yes</i>	<i>flow</i>
<i>v'₂</i>	<i>port</i>	<i>out</i>	<i>yes</i>	<i>pressure</i>
...

TableConnected

<i>port</i>	<i>port</i>
<i>u₂</i>	<i>u'₁</i>
<i>v₂</i>	<i>v'₁</i>

According to the usual approach in MBD, a logical language \mathcal{L}_g is adopted; in particular, $\mathcal{L}_g = (\mathbf{P}, \mathbf{F}, \mathbf{C})$, where **P** is the set of predicates:

$\mathbf{P} = \{\mathbf{comp}(x), \mathbf{ctrdev}(x), \mathbf{port}(x), \mathbf{observable}(x), \mathbf{in}(x), \mathbf{out}(x), \mathbf{flow}(x), \mathbf{pressure}(x), \mathbf{port-of}(x, y), \mathbf{connected}(x, y), \mathbf{controls}(x, y), \mathbf{ok}(x), \mathbf{so}(x), \dots, \mathbf{pipe}(x, u_1, u_2, v_1, v_2), \mathbf{valve}(x, s, u_1, u_2, v_1, v_2)\}$,

\mathbf{F} is the set of functions:

$$\mathbf{F} = \{B_p, B_v, \text{Command}, \Delta\text{FlowValue}, \Delta\text{PressureValue}\},$$

and \mathbf{C} is the set of constants:

$$\mathbf{C} = \{P1, V1, \dots\} \cup \Lambda_{B_p} \cup \dots \cup \{\text{open}, \text{close}, +, -, 0\}.$$

Some of the predicates occurring in \mathbf{P} have been introduced for describing the perception, but some others are suggested by the theory (for instance, **valve** and **pipe**), as it will be clarified later on. In an analogous way, the database also contains tables that provide the semantics of predicates in the language and of formulas in the theory. For instance, two tables $\text{TableValve} = (\text{valve}, \text{ctrdev}, \text{in_flow_port}, \text{out_flow_port}, \text{in_press_port}, \text{out_press_port})$ and $\text{TablePipe} = (\text{pipe}, \text{in_flow_port}, \text{out_flow_port}, \text{in_press_port}, \text{out_press_port})$ are added to \mathcal{D}_g in correspondence to the predicates **valve** and **pipe**.

In our example, the theory contains three types of knowledge: an algebra over the qualitative measurement values $\{+, 0, -\}$, a description of the components, and a set of rules specifying the components behaviors. As an example of the first type of knowledge, the semantics of the qualitative sum \oplus is given:

$$+ \oplus + = +, \quad + \oplus 0 = +, \quad + \oplus - = \{+, 0, -\} \quad 0 \oplus 0 = 0, \quad 0 \oplus - = -, \quad - \oplus - = -$$

The second type of knowledge contains, for instance, the following structural description relating a valve or a pipe to their ports and control devices:

$$\begin{aligned} &\mathbf{valve}(V, s, u_1, u_2, v_1, v_2) \Leftrightarrow \\ &\mathbf{comp}(V) \wedge \mathbf{port}(u_1) \wedge \mathbf{port-of}(u_1, V) \wedge \mathbf{in}(u_1) \wedge \mathbf{flow}(u_1) \wedge \mathbf{port}(u_2) \wedge \mathbf{port-of}(u_2, V) \wedge \\ &\mathbf{out}(u_2) \wedge \mathbf{flow}(u_2) \wedge \mathbf{port}(v_1) \wedge \mathbf{port-of}(v_1, V) \wedge \mathbf{in}(v_1) \wedge \mathbf{pressure}(v_1) \wedge \mathbf{port}(v_2) \wedge \mathbf{port-of}(v_2, V) \wedge \\ &\mathbf{out}(v_2) \wedge \mathbf{pressure}(u_2) \wedge \mathbf{ctrdev}(s) \wedge \mathbf{controls}(s, V) \end{aligned}$$

$$\begin{aligned} &\mathbf{pipe}(P, u_1, u_2, v_1, v_2) \Leftrightarrow \\ &\mathbf{comp}(P) \wedge \mathbf{port}(u_1) \wedge \mathbf{port-of}(u_1, P) \wedge \mathbf{in}(u_1) \wedge \mathbf{flow}(u_1) \wedge \mathbf{port}(u_2) \wedge \mathbf{port-of}(u_2, P) \wedge \\ &\mathbf{out}(u_2) \wedge \mathbf{flow}(u_2) \wedge \mathbf{port}(v_1) \wedge \mathbf{port-of}(v_1, P) \wedge \mathbf{in}(v_1) \wedge \mathbf{pressure}(v_1) \wedge \mathbf{port}(v_2) \wedge \mathbf{port-of}(v_2, P) \wedge \\ &\mathbf{out}(v_2) \wedge \mathbf{pressure}(u_2) \end{aligned}$$

Also the behavioral modes described with equations in Figure 3 can be rewritten in logical terms in the theory. For instance:

$$\begin{aligned} \mathbf{pipe}(P, u_1, u_2, v_1, v_2) \wedge \mathbf{ok}(P) \rightarrow &(\Delta\text{FlowValue}(u_1) = \Delta\text{FlowValue}(u_2)) \wedge \\ &(\Delta\text{PressureValue}(v_1) = \Delta\text{PressureValue}(v_2)) \end{aligned}$$

6 Modelling Abstract Diagnosis with \mathcal{KRA}

In this section we will describe how the \mathcal{KRA} model can be used to formalize an abstraction operation consisting of aggregating pipe $P1$ and valve $V1$ (see Figures 1 and 2), obtaining thus an abstract component $PV1$. To this aim, an operator ω_{ser} is defined at the perception level:

$$\omega_{ser} : \mathbf{PIPE} \times \mathbf{VALVE} \rightarrow \mathbf{SPV}$$

where the new component type **SPV** denotes a serial connection of a pipe and valve.

Notice that the above operator applies to components connected in series; for components connected in parallel, a different operator is to be defined. The operator $\omega_{ser}(P1, V1)$ generates an abstract representation of the fragment of Figure 2 (also reported in the same figure). In the abstraction process the ports connecting $P1$ and $V1$ disappear, whereas the others remain accessible; hiding these ports may imply a reduction in observability, and has the potential of increasing the indiscriminability among diagnoses of the subsystem involving $P1, V1$. We will go back to this issue at the end of this section.

The abstract perception \mathcal{P}_a is now the following one:

- $\mathbf{OBJ}_a = \mathbf{COMP}_a \cup \mathbf{CTRDEV} \cup \mathbf{PORT} \cup \dots$ where $\mathbf{COMP}_a = \mathbf{PIPE} \cup \mathbf{VALVE} \cup \mathbf{SPV}$
- $\mathbf{ATT}_a = \mathbf{ATT}_g - \{ObjType_g\} \cup \{ObjType_a : \mathbf{COMP}_a \rightarrow \Lambda_{ObjType_g} \cup \{spv\}\}$
- $\mathbf{FUNC}_a = \mathbf{FUNC}_g \cup \{B_{SPV} : \mathbf{SPV} \rightarrow C_{SPV}\}$
- $\mathbf{REL}_a = \{ \mathbf{port-of} \subseteq \mathbf{PORT} \times \mathbf{COMP}_a, \mathbf{connected} \subseteq \mathbf{PORT} \times \mathbf{PORT}, \mathbf{controls} \subseteq \mathbf{CTRDEV} \times \mathbf{COMP}_a \}$

Notice that in the abstract world the object types **PIPE** and **VALVE** are not deleted, because, in the abstract system, non aggregated pipes and valves still appear. Only, the new type **SPV** is added. The set of observations becomes then:

$\mathbf{OBS}_a = \{ (PV1, s_1, u_1, v_1, u'_2, v'_2, \dots), (ObjType(PV1) = spv, ObjType(u_1) = port, \dots), (Direction(u_1) = in, Direction(v_1) = in, \dots), (Observable(u_1) = yes, \dots), (MeasureType(u_1) = flow, \dots), Command(s_1) = open, (\Delta FlowValue(u_1) = +, \Delta PressureValue(v_1) = +, \dots), (\mathbf{port-of}(u_1, PV1), \dots), \mathbf{controls}(s_1, PV1) \}$ Starting from the definition of the aggregation operator at the perception level, the corresponding $\delta_{ser}(\mathcal{D}_g)$ generates \mathcal{D}_a from the tables in \mathcal{D}_g , by performing the following operations:

- Any row of $\mathbf{TableObj}_g$ in which either $P1$ or $V1$ appears is removed, and a row with the object $PV1$ is added, generating thus $\mathbf{TableObj}_a$.
- Any row in any table of \mathcal{D}_g , in which either $P1$ or $V1$ appears, is deleted using relational algebra operators, as well as the ports that connect $P1$ with $V1$ (these are found in $\mathbf{TablePortOf}_g$). External ports of $P1$ and $V1$ are attached to $PV1$.
- Control devices acting either on $P1$ or on $V1$ shall act upon $PV1$.

All the above operations can be formalized and executed automatically through application of relational algebra operators to \mathcal{D}_g .

At the language level, as no object type disappears, there are no changes, except that a new predicate SPV and a new constant pvl are added. Also new predicates corresponding to the behavioral modes of an abstract component of

type PV need to be added; they are derived as a result of the abstraction at the theory level (see below). More precisely, in $\mathcal{L}_a = (\mathbf{P}_a, \mathbf{F}_a, \mathbf{C}_a)$ we have: $\mathbf{P}_a = \mathbf{P}_g \cup \{\mathbf{spv}(x), \mathbf{oam}_1(x), \dots\}$, $\mathbf{F}_a = \mathbf{F}_g \cup \{B_{spv}\}$, and $\mathbf{C}_a = \mathbf{C}_s \cup \{PV1\} \cup \Lambda_{B_{spv}}$.

At the theory level, formulas describing the structure and behavior of **SPV** components must be synthesized and added to the abstract theory \mathcal{T}_a . In particular:

$$\mathbf{spv}(PV, s, u_1, v_1, u'_2, v'_2) \Leftrightarrow \mathbf{valve}(V, s, u'_1, u'_2, v'_1, v'_2) \wedge \mathbf{pipe}(P, u_1, u_2, v_1, v_2) \wedge \mathbf{connected}(u_2, u'_1) \wedge \mathbf{connected}(v_2, v'_1) \wedge \mathbf{crtdev}(s, PV) \wedge \mathbf{port-of}(u_1, PV) \wedge \mathbf{port-of}(v_1, PV) \wedge \mathbf{port-of}(u'_2, PV) \wedge \mathbf{port-of}(v'_2, PV)$$

Once we have defined the structure of the abstract component **SPV**, we need to compute its behavioural modes B_{spv} starting from B_p and B_v . This computation can be done automatically, by combining the behaviors of **PIPE** and **VALVE** and collecting the combinations that results in the same abstract patterns, which are exactly the indiscriminable states (according to Definition 5) of a subsystem composed by a pipe and a valve connected in series. To each one of these combinations a new name (denoting a behavior of the abstract component) is assigned, and, correspondingly, a new predicate is added to \mathbf{P}_a in \mathcal{L}_a . For **SPV**, we obtain:

$$\begin{aligned} Oam_1 &\leftrightarrow \{B_p(P1) = ok, B_v(V1) = ok\}, \{B_p(P1) = ok, B_v(V1) = so\} \\ Oam_2 &\leftrightarrow \{B_p(P1) = pc, B_v(V1) = ok\}, \{B_p(P1) = pc, B_v(V1) = so\} \\ Oam_3 &\leftrightarrow \{B_p(P1) = br, B_v(V1) = ok\}, \{B_p(P1) = br, B_v(V1) = so\}, \\ &\quad \{B_p(P1) = br, B_v(V1) = sc\} \\ Oam_4 &\leftrightarrow \{B_p(P1) = ok, B_v(V1) = sc\}, \{B_p(P1) = pc, B_v(V1) = sc\}, \\ &\quad \{B_p(P1) = cl, ok(V1)\}, \{B_p(P1) = cl, B_v(V1) = so\}, \\ &\quad \{B_p(P1) = cl, B_v(V1) = sc\} \end{aligned}$$

Starting from a pipe and a valve serially connected, in principle the abstract component could have 12 different behavioral modes. Actually, a large number of behavioral assignments to the components $P1$ and $V1$ collapse into the same abstract mode of the abstract component $PV1$; this is a strong indication that the abstraction is not only possible but really useful.

Providing the system with only the definition of ω_{ser} , the $\mathcal{KR}\mathcal{A}$ models allows the needed transformations to be done automatically at all levels, both concerning the structure and concerning the behaviors.

The application of ω_{ser} as defined above may hide some measurements on the internal ports of the pipe-valve subsystems to which it is applied; in such cases it can be shown that, by performing diagnosis with the abstract model, some spurious diagnoses may be generated, i.e. diagnoses that would have been ruled out by reasoning at the ground level.

Although space prevents us from giving details here, it turns out that adding a simple constraint on ω_{ser} guarantees that no loss of discrimination power can happen when performing diagnostic reasoning at the abstract level; in particular, it is sufficient to require that ω_{ser} is applicable only when none of the internal ports of the pipe-valve subsystem is observable.

$s_1 = \text{open}$	
$Oam_1(pv_1)$	$\Delta f_{out} = \Delta f_{in} ; \Delta r_{in} = \Delta r_{out}$
$Oam_2(pv_1)$	$\Delta f_{out} = \Delta f_{in} ; \Delta r_{in} = \Delta r_{out} \oplus +$
$Oam_3(pv_1)$	$\Delta f_{out} = - ; \Delta r_{in} = -$
$Oam_4(pv_1)$	$\Delta f_{out} = \Delta f_{in} = - ; \Delta r_{in} = +$

Fig. 5. Models of Abstract Component pv_1 when s_1 is set to open

7 A Test of the Approach

In this section we will briefly describe how the approach based on \mathcal{KRA} is able to perform a number of different abstractions starting from the ground model of the hydraulic system introduced in Figure 1. As we have seen in the previous section, the application of the aggregation operator ω_{ser} to $P1$ and $V1$ produces an abstract component $PV1$ of type **SPV** with 4 behavioral modes when the command s_1 is set to *open*, (see Figure 5). The resulting abstract hydraulic system is reported in Figure 4.

By aggregating the abstract component $PV1$ with $P2$, we get a new abstract component $PVP1$, characterized by only 5 behavioral modes (again when the command s_1 is set to *open*). In a similar way, by aggregating $P3$ with $V2$, and, then, the resulting abstract component $PV2$ with $P4$, we get an abstract component $PVP2$, again with 5 behavioral modes when the command s_2 is set to *close*. The abstract hydraulic system resulting from these abstraction steps is depicted in the upper portion of Figure 6. The final abstraction step involves the aggregation of the *SPLIT*, the *JOIN* and the abstract components $PVP1$ and $PVP2$ through the aggregation operator ω_{par} which aggregates components connected in parallel. The resulting abstract component $SJ1$ is characterized by only 9 behavioral modes (reported in Figure 7) when s_1 is set to *open* and s_2 is set to *close*, just a small fraction of the potential $48 \cdot 48$ different behavioral

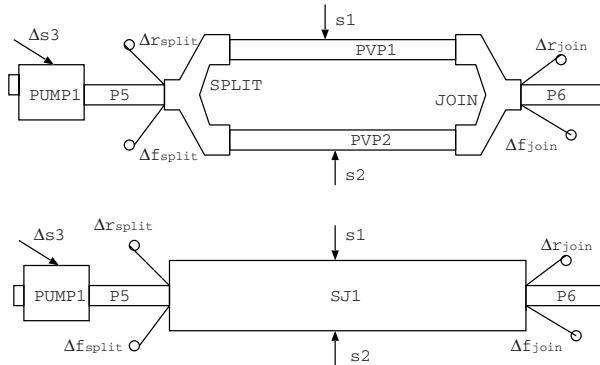


Fig. 6. The Hydraulic System after the application of aggregation operators in series (upper part) and the application of ω_{par}

$am_1(SJ1)$	$\Delta f_{out} = \Delta f_{in} ; \Delta r_{in} = \Delta r_{out}$
$am_2(SJ1)$	$\Delta f_{out} = \Delta f_{in} ; \Delta r_{in} = \Delta r_{out} \oplus +$
$am_3(SJ1)$	$\Delta f_{out} = - ; \Delta r_{in} = -$
$am_4(SJ1)$	$\Delta f_{out} = \Delta f_{in} = - ; \Delta r_{in} = +$
$am_5(SJ1)$	$\Delta f_{out} = -$
$am_6(SJ1)$	$\Delta r_{in} = \Delta r_{out} = -$
$am_7(SJ1)$	$\Delta f_{out} = \Delta f_{in} ; \Delta r_{in} = \Delta r_{out} = +$
$am_8(SJ1)$	$\Delta r_{in} = \Delta r_{out}$
$am_9(SJ1)$	$\Delta f_{out} = \Delta f_{in} ; \Delta r_{in} = +$

Fig. 7. Model of the Abstract Component $SJ1$

modes that can be obtained by considering the cross product of the behavioral modes of the components involved in the abstraction.

Note that in this example we have applied ω_{ser} (and ω_{par}) only to subsystems whose internal ports were non-observable. In this way we adhere to the constraints discussed at the end of the previous section and therefore we can safely use the abstract model for diagnostic reasoning without any risk of finding additional (spurious) diagnoses w.r.t. the ones we would find by reasoning at the ground level.

8 Conclusions

In this paper we have shown how the process of Model-Based Diagnosis can profit from a formalization of the abstraction process. In recent years, new approaches have been proposed in MBD to exploit the degree of observability for defining useful abstractions (e.g., [10]), while other works have started to investigate the problem of automatic synthesis of abstract models (e.g., [9], [11]). While these approaches have developed some interesting solutions to the problem of abstracting models for MBD, they have failed to investigate the relations between proposed methods and general theories of abstraction. The present work represents a step in the direction of filling this gap. In particular, the present paper describes how the aggregation of components in MBD can be captured within the \mathcal{KRA} model, which has been originally introduced for modelling abstraction in concept representation tasks. This shows that the \mathcal{KRA} model is general and flexible enough for taking into consideration the requirements and the characteristics of the diagnostic task.

On the other hand the \mathcal{KRA} model offers a number of advantages in capturing the process of building abstract models for MBD. In particular, the possibility of defining aggregation operators at the perception level allows the automatic synthesis of abstract models and, by taking into account the observability of the system to be diagnosed, it is possible to preserve the diagnostic discrimination power when moving from the ground to the abstract model. This is an important step since it allows the behaviour of new complex abstract components to be automatically learned, as sketched in the example of the hydraulic system. In addition to the abstraction at the perception level, the model addresses the problem of automatically extending the representation changes to all levels needed

to perform the diagnostic tasks, namely the theory, the language and the databases. Most previous works on modelling abstraction in MBD have focused on either the language or the semantic levels, seldom considering their interactions and never their compatibility with the observations from the external world.

References

1. Holte, R., Mkadmi, T., Zimmer, R., MacDonald, A.: Speeding up problem-solving by abstraction: A graph-oriented approach. *Art. Intelligence* 85, 321–361 (1996)
2. Knoblock, C., Tenenber, J., Qiang, Y.: A spectrum of abstraction hierarchies for planning. In: *Proc. AAAI WS on AGAA*, pp. 24–35 (1990)
3. Mozetič, I.: Hierarchical model-based diagnosis. *Int. Journal of Man-Machine Studies* 35(3), 329–362 (1991)
4. Subramanian, D.: Automation of abstractions and approximations: Some challenges. In: *Proc. AAAI WS on AGAA*, pp. 76–77 (1990)
5. Giunchiglia, F., Walsh, T.: A theory of abstraction. *Art. Intell.* 57(2-3), 323–389 (1992)
6. Nayak, P., Levy, A.: A semantic theory of abstraction. In: *Proc. IJCAI*, pp. 196–202 (1995)
7. Korf, R.: Toward a model of representation changes. *Art. Intell.* 14, 41–78 (1980)
8. Saitta, L., Zucker, J.: Semantic abstraction for concept representation and learning. In: *Proc. SARA*, pp. 103–120 (1998)
9. Sachenbacher, M., Struss, P.: Task-dependent qualitative domain abstraction. *Art. Intell.* 162(1-2), 121–143 (2005)
10. Chittaro, L., Ranon, R.: Hierarchical model-based diagnosis based on structural abstraction. *Art. Intell.* 155(1-2), 147–182 (2004)
11. Torta, G., Torasso, P.: Automatic abstraction in component-based diagnosis driven by system observability. In: *Proc. IJCAI*, pp. 394–400 (2003)
12. Struss, P., Malik, A., Sachenbacher, M.: Qualitative modeling is the key to automated diagnosis. In: *Proc. IFAC96* (1996)
13. Saitta, L., Zucker, J.D.: Abstraction and complexity measures. In: *Proc. SARA-2007* (2007)
14. Plaisted, D.: Theorem proving with abstraction. *Art. Intelligence* 16, 47–108 (1981)
15. Tenenber, J.: Preserving consistency across abstraction mappings. In: *Proc. IJCAI 1987*, pp. 1011–1014 (1987)

Boolean Approximation Revisited

Peter Schachte* and Harald Søndergaard

*NICTA Victoria Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Vic. 3010, Australia
{schachte,harald}@csse.unimelb.edu.au

Abstract. Most work to date on Boolean approximation assumes that Boolean functions are represented by formulas in conjunctive normal form. That assumption is appropriate for the classical applications of Boolean approximation but potentially limits wider use. We revisit, in a lattice-theoretic setting, so-called envelopes and cores in propositional logic, identifying them with upper and lower closure operators, respectively. This leads to recursive representation-independent characterisations of Boolean approximation for a large class of classes. We show that Boolean development can be applied in a representation-independent setting to develop approximation algorithms for a broad range of Boolean classes, including Horn and Krom functions.

1 Introduction

Since the seminal work by Selman and Kautz [22] there has been considerable interest in Horn approximations of propositional formulas. The original motivation for Horn approximation was the fact that it could allow faster query answering with propositional knowledge bases. But concepts of Boolean function “approximation” and “abstraction” are found in other fields of computer science, including computational learning, symbolic problem-solving, property testing and program analysis. A typical task in any of these may be to find the best, say, monomial, monotone, or Horn theory supporting a given set of data, or “covering” a given theory.

Selman and Kautz’s idea of querying (and performing deductions from) upper and lower Horn approximations of a knowledge-base has subsequently been adapted and extended in various directions, and additional uses of Horn approximation have been suggested. Most notable is the recent contribution by del Val [8]. Del Val shows that Kautz and Selman’s Horn envelope algorithm carries over to all Boolean function classes closed under subsumption and he proposes an improved algorithm that is applicable if, additionally, the complement of a class is closed under resolution. Moreover, del Val discusses the case of first-order predicate logic, showing how the original concepts can be extended in this direction too. Note that the concepts that are central to del Val [8], such as closure under subsumption and resolution, reflect the underlying assumption of clausal-form representation.

Zanuttini [24,25] discusses the use of other classes of Boolean functions for approximation, in particular affine functions, and also the use of approximations in the setting of abduction. It is argued that affine approximations have certain advantages over Horn approximations, most notably the fact that they do not blow out in size. (Note, however, that the affine functions are very unevenly distributed across the lattice of Boolean functions, having only sets of models whose cardinality is 2^n for some n . Hence with affine approximation, roughly speaking the “weakest half” of the Boolean functions are *all* approximated to the vacuous function “true”.) Zanuttini [24] proves that the affine envelope of a relation $R \in \{0, 1\}^n$ can be computed in time $O(|R|n^3 + n^4)$. He recalls a result from Dechter and Pearl [7] that the Krom envelope can be computed in time $O(|R|n^2)$.

There have been proposals for representations other than clausal form, such as characteristic models [13]. Horiyama and Ibaraki [11] suggest the use of ROBDDs for knowledge bases and give algorithms to recognise unate and Horn functions represented as ROBDDs. Schachte and Søndergaard [20] give algorithms for the approximation of Boolean functions represented as ROBDDs, covering several classes, including monotone and Horn functions. Khardon [15] and Horiyama and Ibaraki [12] are concerned with the translation between different representations and establish many interesting results.

A large body of work (see for example Cadoli and Scarcello [3]) is primarily interested in the problem of finding maximal lower Horn approximations. While the results in this paper also apply to lower approximations, we are only interested in the case where approximations are unique, and so we make no contribution to the particular discussion about lower Horn bounds.

A large variety of special classes of Boolean functions, including Horn, are used in program analysis, to automatically reason about runtime properties of programs. In all kinds of static program analysis, approximation plays a pivotal role. The runtime properties of interest are almost always undecidable, so reasoning is necessarily approximate, and abstraction is therefore integral to the definition of a program analysis. Boolean approximation, in the sense of calculating the strongest logical consequence, in some class, of a given Boolean function, is used in at least two different ways. One is to accelerate convergence of the analysis via so-called widening [6]. The other is where approximation finds a role in basic operations on “runtime state descriptions”, as it happens in set-sharing analysis for logic programs. In one view [4] this analysis uses positive Boolean functions (those that evaluate to true when all arguments are true) to express how the instantiation of one variable may affect other variables. For example, the formula $x \leftrightarrow y$ would express the constraint that any goal that further instantiated program variable x would necessarily further instantiate y .

The ubiquity of applications of propositional logic, together with the fact that concepts of Boolean function approximation are found in many different fields of computer science, suggests that it may be fruitful to revisit the approximation problem outside the context of clausal-form representations. In this paper we consider representation-independent aspects of approximation, as well as al-

gorithms for Boolean approximation that can use a variety of data structures to represent Boolean functions. We view the approximation problem under a lattice-theoretic lens and suggest a general approach to finding approximation algorithms for an important class of classes.

We assume the reader is familiar with propositional logic and elementary lattice and order theory. Section 2 gives relevant definitions and introduces some Boolean function classes of interest. In Section 3 we discuss the view of Boolean classes as closure operators more formally. Section 4 introduces a class of classes, for which a general approach to finding approximations is possible, and we explain the approach. In Section 5 we instantiate the general characterisation to different classes, including Horn, Krom, monotone and antitone functions. Section 6 concludes.

2 Preliminaries: Boolean Functions

Let $\mathcal{B} = \{0, 1\}$ and let \mathcal{V} be a countably enumerable set of variables. A *valuation* $\mu : \mathcal{V} \rightarrow \mathcal{B}$ is an assignment of truth values to the variables in \mathcal{V} . Let $\mathcal{I} = \mathcal{V} \rightarrow \mathcal{B}$ denote the set of \mathcal{V} -valuations.

A Boolean function over \mathcal{V} is a function $\varphi : \mathcal{I} \rightarrow \mathcal{B}$. We let \mathbf{B} denote the set of all Boolean functions over \mathcal{V} . The ordering on \mathcal{B} is the usual: $x \leq y$ iff $x = 0 \vee y = 1$. \mathbf{B} is ordered pointwise, so that the ordering relation corresponds exactly to classical entailment, \models . It is convenient to overload the symbols for truth and falsehood. Thus we let 1 denote the largest element of \mathbf{B} (that is, $\lambda\mu.1$) as well as of \mathcal{B} . Similarly 0 also denotes the smallest element of \mathbf{B} (that is, $\lambda\mu.0$) as well as of \mathcal{B} .

A valuation μ is a *model* for φ , denoted $\mu \models \varphi$, if $\varphi(\mu) = 1$. We use the notation $\mu[x \mapsto i]$, where $x \in \mathcal{V}$ and $i \in \mathcal{B}$, to denote the valuation μ updated to map x to i , that is,

$$\mu[x \mapsto i](v) = \begin{cases} i & \text{if } v = x \\ \mu(v) & \text{otherwise} \end{cases}$$

Also, to facilitate a definition (in Section 4) of “unbiased” Boolean function classes, that is, classes defined without reference to any *specific* variables, we define the concept of “swapping” variables in a valuation:

$$\mu_{[x \leftrightarrow y]}(v) = \begin{cases} \mu(y) & \text{if } v = x \\ \mu(x) & \text{if } v = y \\ \mu(v) & \text{otherwise} \end{cases}$$

We lift this to apply to Boolean functions by defining $\varphi_{[x \leftrightarrow y]}(\mu) = \varphi(\mu_{[x \leftrightarrow y]})$. That is, $\varphi_{[x \leftrightarrow y]}$ simultaneously replaces all occurrences of x in φ with y and occurrences of y with x .

For $\varphi \in \mathbf{B}$ we use $\overline{\varphi}$ to denote φ 's negation. Let $\overline{\mathcal{V}} = \{\overline{x} \mid x \in \mathcal{V}\}$ be the set of negated variables. A *literal* is a member of the set $\mathcal{V} \cup \overline{\mathcal{V}}$, that is, a variable or a negated variable. We use φ_x^i to stand for φ with x instantiated to i , that

is, $\varphi_x^i(\mu) = \varphi(\mu[x \mapsto i])$. We say that φ is *independent* of literal x (and also of literal \bar{x}) when $\varphi_x^0 = \varphi_x^1 = \varphi$, and we write this $\varphi \approx x$. We say that φ *depends* on x iff φ is not independent of x .

The *dual* of a Boolean function φ is the function that is obtained by interchanging the roles of the truth values 0 and 1 . A simple way of turning a formula for φ into a formula for φ 's dual is to change the sign of every literal in φ and negate the whole resulting formula. For example, the dual of $x \wedge (\bar{y} \vee z)$ is $x \vee (\bar{y} \wedge z)$ — De Morgan's laws can be regarded as duality laws.

Define $\tilde{\varphi}$ as the dual of φ . Following Halmos [10], we call $\tilde{\varphi}$ the *contra-dual* of φ . Clearly, given a formula for φ , a formula for $\tilde{\varphi}$ is obtained by changing the sign of each literal in φ . As an example, if $\varphi = (x \leftrightarrow y) \rightarrow z$ then $\tilde{\varphi} = (x \leftrightarrow y) \rightarrow \bar{z}$. Given a truth table for a Boolean function, the truth table for its contra-dual is obtained by turning the result column upside down. The mapping $\varphi \mapsto \tilde{\varphi}$ is an involution, and monotone: $\psi \models \varphi$ iff $\tilde{\psi} \models \tilde{\varphi}$. For any class $\Delta \subseteq \mathbf{B}$, we let $\tilde{\Delta}$ denote the class $\{\tilde{\varphi} \mid \varphi \in \Delta\}$.

Function classes Δ central to this paper include:

- H:** A *Horn* function is one whose set of models is closed under pointwise conjunction. That is, **H** (and only **H**) functions φ satisfy the requirement that for all valuations μ and μ' , if $\mu \models \varphi$ and $\mu' \models \varphi$, then $\mu \wedge \mu' \models \varphi$. **H** is the set of functions that can be written in conjunctive normal form $\bigwedge(\ell_1 \vee \dots \vee \ell_n)$, $n \geq 0$, with at most one positive literal ℓ per clause.
- $\tilde{\mathbf{H}}$: A *contra-dual Horn* function φ satisfies the requirement that for all valuations μ and μ' , if $\mu \models \varphi$ and $\mu' \models \varphi$, then $\mu \vee \mu' \models \varphi$. A $\tilde{\mathbf{H}}$ function can be written in CNF with each clause containing at most one negative literal.
- M:** A *monotone* function φ satisfies the requirement that for all valuations μ and μ' , $\mu \vee \mu' \models \varphi$ when $\mu \models \varphi$. Here \vee denotes pointwise disjunction. Monotone functions are sometimes referred to as *isotone*. Syntactically the class is most conveniently described as the functions generated by $\{\wedge, \vee, 0, 1\}$, see for example Rudeanu's [19] Theorem 11.3.
- $\tilde{\mathbf{M}}$: An *antitone* function φ has the property that, for all valuations μ and μ' , if $\mu \models \varphi$ then $\mu \wedge \mu' \models \varphi$.
- K:** A *Krom* function is one whose set of models is closed under pointwise application of the majority-of-3 function $\lambda x, y, z. (x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$. It is generated by formulas in CNF with a most two literals per clause, and its members are also referred to as 2-CNF or *bijunctive*.
- L:** This is the class 1-CNF consisting of functions that can be written as conjunctions of single-literal (or empty) clauses.
- S:** This is the intersection of **H** and $\tilde{\mathbf{H}}$, that is, **L** extended with simple dependencies of the form $x \rightarrow y$.
- V:** This is **L** restricted to positive literals.
- $\tilde{\mathbf{V}}$: This is **L** restricted to negative literals.
- P:** A *positive* function is one that is satisfied by the unit valuation $\lambda v. 1$.
- D:** A *definite* function is one that is both positive and Horn.

C: This class consists of the constant functions 0 and 1 .

1: This is the class consisting of the constant function 1 only.

All of these classes, apart from **P**, **D**, and **1**, contain 0 . It is immediate that $\mathbf{M} \subseteq \tilde{\mathbf{H}}$ and $\tilde{\mathbf{M}} \subseteq \mathbf{H}$. Figure 1 shows the classes as a Hasse diagram, ordered by the subset ordering.

These classes find widespread use in computer science and several play central roles in the theory of propositional expressiveness, in computational complexity theory, or both. **M** and **P** are the classes “A:a” and “ β ” of Post’s functional completeness result [18] (made more accessible by Pelletier and Martin [17]), Post’s remaining three classes being the dual of **P**, the alternating functions, and the self-dual functions. Schaefer’s celebrated SAT dichotomy result [21] makes use of six classes, five of which are: **H** and $\tilde{\mathbf{H}}$ (called “weakly negative” and “weakly positive” respectively), **P** and its contra-dual (“1-valid” and “0-valid”), and **K** (or “bijunctive”). The sixth is the class of affine functions.

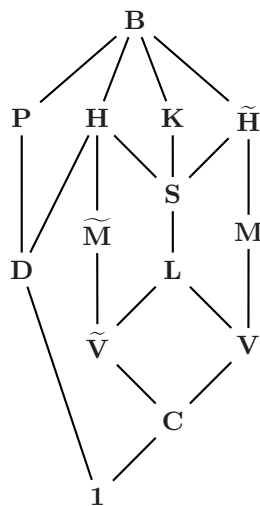


Fig. 1. Boolean function classes

3 Approximation as Closure Operators

We are interested in the problem of approximating a Boolean function φ , in the sense of finding, when it exists, the strongest function ψ from a given class Δ , entailed by φ . This approximation is sometimes referred to as the “ Δ envelope” of φ [14]. We denote this $\Delta_{\uparrow}(\varphi)$. Also of interest, for some classes Δ , is the weakest function $\psi \in \Delta$ which entails φ . Such a ψ is sometimes referred to as the “ Δ core” of φ , denoted $\Delta_{\downarrow}(\varphi)$.

What are the essential properties of an approximation operator ρ , whether it produces envelopes or cores? A first natural requirement is that it is *idempotent*, that is, $\rho(\varphi) = \rho(\rho(\varphi))$ for all $\varphi \in \mathbf{B}$. In other words, ρ acts instantaneously and is the identity function on the set of approximations $\rho(\mathbf{B})$. A second natural requirement is that it is *monotone*, that is, $\varphi \models \varphi'$ implies $\rho(\varphi) \models \rho(\varphi')$ for all Boolean functions φ and φ' . In other words, ρ preserves entailment and thus does not squander information. Together the two requirements say that ρ is a *retraction*.

The only requirement remaining is the one that provides a direction for the approximation. An *upper* approximation operator (yielding “envelopes”) is *extensive*, that is, $\varphi \models \rho(\varphi)$ for all φ . A *lower* approximation operator (yielding

“cores”) is *reductive*, that is, $\rho(\varphi) \models \varphi$ for all φ . An extensive retraction is known as an *upper closure operator*, or uco, and a reductive retraction is a *lower closure operator*, or lco. Retractions exists that are neither upper nor lower approximation operators, but they are not of interest here.

All these concepts are well known in lattice and order theory [1]. The definition of approximation operators make sense for operators defined on lattices more generally— \mathbf{B} is just a special case.

3.1 Upper Closure Operators

General properties of closure operators [5,23] hold for \mathbf{B} . If $\rho : \mathbf{B} \rightarrow \mathbf{B}$ is a uco then $\rho(\mathbf{B})$ is a (complete) lattice with least element $\rho(\theta)$, greatest element 1 , greatest lower bound operation \bigwedge , and least upper bound operation $\lambda S.\rho(\bigvee S)$. It is a sublattice of L if and only if ρ is additive, that is, $\rho(\bigvee S) = \bigvee \rho(S)$ for all $S \subseteq \mathbf{B}$. In any case,

$$\rho(\bigwedge S) \models \bigwedge \rho(S) = \rho(\bigwedge \rho(S)) \tag{1}$$

$$\bigvee \rho(S) \models \rho(\bigvee S) = \rho(\bigvee \rho(S)) \tag{2}$$

It follows that $\rho(\mathbf{B})$ always contains 1 and is closed under conjunction. Conversely, any $\Delta \subseteq \mathbf{B}$ containing 1 and closed under conjunction uniquely determines a uco, defined by

$$\Delta_{\uparrow}(\varphi) = \bigwedge \{\psi \mid \psi \in \Delta \text{ and } \varphi \models \psi\}$$

A family of ucos $\{\exists_v\}_{v \in Var}$ is given by existential quantification on \mathbf{B} : $\exists_x = \lambda\varphi. \exists x.\varphi$ is a uco, as is easily verified.

Given two upper closure operators ρ and ρ' on \mathbf{B} , $\rho \circ \rho'$ need not be an upper closure operator. For example, with the uco ρ defined by

$$\rho(\varphi) = \begin{cases} x & \text{if } \varphi \models x \\ 1 & \text{otherwise} \end{cases}$$

$\rho \circ \exists_x$ is not idempotent, as $\rho(\exists_x(\theta)) = x \neq 1 = \exists_x(\rho(x))$. However, if $\rho \circ \rho' = \rho' \circ \rho$ then the composition is also an upper closure operator, and $\rho(\rho'(\mathbf{B})) = \rho'(\rho(\mathbf{B})) = \rho(\mathbf{B}) \cap \rho'(\mathbf{B})$ [9,16].

Proposition 1. Let Δ_{\uparrow} be a uco on \mathbf{B} . If $\exists_x \circ \Delta_{\uparrow} = \Delta_{\uparrow} \circ \exists_x$ then Δ is closed under \exists_x .

Proof: Assume that $\exists_x \circ \Delta_{\uparrow} = \Delta_{\uparrow} \circ \exists_x$. For $\varphi \in \Delta$ we have $\exists_x(\varphi) = \exists_x(\Delta_{\uparrow}(\varphi)) = \Delta_{\uparrow}(\exists_x(\varphi))$. Hence $\exists_x(\varphi)$ is in Δ . ■

¹ These consequences are also easy to show directly: Since $1 \models \rho(1)$, $\rho(1) = 1$. Moreover, by monotonicity, $\rho(\varphi \wedge \varphi')$ entails both $\rho(\varphi)$ and $\rho(\varphi')$, and so $\rho(\varphi \wedge \varphi') \models \rho(\varphi) \wedge \rho(\varphi')$. If φ and φ' are fixed points for ρ then the last statement reduces to $\rho(\varphi \wedge \varphi') \models \varphi \wedge \varphi'$. Since $\varphi \wedge \varphi' \models \rho(\varphi \wedge \varphi')$, $\varphi \wedge \varphi'$ is also a fixed point. In other words, if φ and φ' are in $\rho(\mathbf{B})$, so is $\varphi \wedge \varphi'$.

The converse does not hold. Take \mathbf{P} , that class of Boolean functions satisfied by the unit valuation λv . 1. As $\exists_x(\varphi) = \varphi_x^0 \vee \varphi_x^1$, and φ_x^1 is in \mathbf{P} whenever φ is, \mathbf{P} is closed under \exists_x . However,

$$\exists_x(\mathbf{P}_\uparrow(\theta)) = \exists_x(\bigwedge_{v \in \mathcal{V}} v) = \bigwedge_{v \in \mathcal{V} \setminus \{x\}} v \neq \bigwedge_{v \in \mathcal{V}} v = \mathbf{P}_\uparrow(\theta) = \mathbf{P}_\uparrow(\exists_x(\theta))$$

Also note that $\exists_x \circ \Delta_\uparrow = \Delta_\uparrow \circ \exists_x$ does not imply closure under instantiation. The uco induced by $\mathbf{P} \cup \{\theta\}$ has the former property, but is not closed under instantiation—for example, $(y \rightarrow x)_x^0 = \bar{y}$.

3.2 Lower Closure Operators

We can develop analogous results for lower closure operators. We shall not do that in detail, but note that a class of Boolean functions induced by an lco contains θ and is closed under disjunction. Conversely, any $\Delta \subseteq \mathbf{B}$ containing θ and closed under disjunction uniquely determines an lco

$$\Delta_\downarrow(\varphi) = \bigvee \{ \psi \mid \psi \in \Delta \text{ and } \psi \models \varphi \}$$

A family of lcos is given by universal quantification, namely $\lambda\varphi . \forall x. \varphi$ is an lco.

3.3 Boolean Development

The characterisations of envelopes that we develop in the next section have come about by considering how closure operators can be applied to Boolean functions expressed through Boolean development, that is, the principle² that

$$\varphi = (\bar{x} \wedge \varphi_x^0) \vee (x \wedge \varphi_x^1) \tag{3}$$

or, by duality,

$$\varphi = (\bar{x} \vee \varphi_x^1) \wedge (x \vee \varphi_x^0) \tag{4}$$

The latter form proves more useful in the context of upper closure operators.

4 Computing Approximations

The approximation techniques presented in this paper apply to a broad range of Boolean classes. However, some restrictions must be imposed to permit the approach to work.

4.1 Decomposable and Unbiased Classes

Definition 1. We say a set $\Delta \subseteq \mathbf{B}$ is *unbiased* iff for all $\psi \in \Delta$ and variables $x, y \in \mathcal{V}$, $\psi_{[x \rightleftharpoons y]} \in \Delta$. ■

² The principle, also known as Shannon expansion, goes back to Boole, albeit in the equivalent form $\varphi = (\bar{x} \wedge \varphi_x^0) + (x \wedge \varphi_x^1)$ where ‘+’ denotes exclusive or.

Thus an unbiased class does not treat any variable differently than any other. An example of a class that is not unbiased is the class of functions that entail y . However, all the usual Boolean classes are unbiased, and restricting our attention to unbiased classes is not a significant limitation on the applicability of our approach. “Unbiased” and “closed under existential quantification” are independent concepts: neither entails the other or its negation.

Definition 2. We say a class $\Delta \subseteq \mathbf{B}$ is *decomposable* iff for any $\varphi, \psi \in \mathbf{B}$ and variable $x \in \mathcal{V}$, if $(x \vee \varphi) \wedge (\bar{x} \vee \psi) \in \Delta$ then $x \vee \varphi \in \Delta$ and $\bar{x} \vee \psi \in \Delta$. ■

Most well-known classes of Boolean functions are decomposable. For example, \mathbf{P} , \mathbf{H} , \mathbf{M} , and \mathbf{K} are decomposable. Some classes, however, are not decomposable. Section 2 mentioned the alternating and affine classes, and these fall outside the scope of our method. To see that the alternating, and hence affine, classes are not decomposable, note that $(x \vee y) \wedge (\bar{x} \vee \bar{y})$ is alternating, but neither conjunct is.

4.2 Quotient Classes

In the following definitions, we shall make use of certain classes, which we call quotient classes, related to the class to which we want to approximate. We shall see that, if we can approximate to a class’s quotient classes, we can approximate to the class. Happily, a class’s quotient classes are generally easier to approximate to than the class itself, as will be seen in Section 5.

Definition 3. For each class $\Delta \subseteq \mathbf{B}$ we define the following quotient classes:

$$\begin{aligned} \Delta^\vee &= \{\psi \mid \text{for all } x \in \mathcal{V} \text{ with } \psi \approx x, (x \vee \psi) \in \Delta\} \\ \Delta^{\bar{\vee}} &= \{\psi \mid \text{for all } x \in \mathcal{V} \text{ with } \psi \approx x, (\bar{x} \vee \psi) \in \Delta\} \\ \Delta^\wedge &= \{\psi \mid \text{for all } x \in \mathcal{V} \text{ with } \psi \approx x, (x \wedge \psi) \in \Delta\} \\ \Delta^{\bar{\wedge}} &= \{\psi \mid \text{for all } x \in \mathcal{V} \text{ with } \psi \approx x, (\bar{x} \wedge \psi) \in \Delta\} \\ \Delta^{\mathbf{C}} &= \Delta \cap \mathbf{C} \end{aligned}$$

Note that $\Delta^{\mathbf{C}}$ is $\mathbf{0}$, $\mathbf{1}$, or \mathbf{C} , according as \emptyset , $\mathbf{1}$, or both are in \mathbf{C} . The next results shows that a quotient class is a closure operator when the original class is.

Proposition 2. For any $\Delta \subseteq \mathbf{B}$, if Δ is closed under conjunction and includes $\mathbf{1}$, then the same is true of Δ^\vee , $\Delta^{\bar{\vee}}$, and $\Delta^{\mathbf{C}}$. Similarly, if $\emptyset \in \Delta$ and Δ is closed under disjunction, then the same is true of Δ^\wedge , $\Delta^{\bar{\wedge}}$, and $\Delta^{\mathbf{C}}$.

Proof: Both claims trivially hold for $\Delta^{\mathbf{C}}$, and $\mathbf{1} \in \Delta^\vee$, $\mathbf{1} \in \Delta^{\bar{\vee}}$, $\emptyset \in \Delta^\wedge$, and $\emptyset \in \Delta^{\bar{\wedge}}$ by construction.

We prove Δ^\vee is closed under conjunction when Δ is; the proof for the other classes is similar. Let $\Delta \subseteq \mathbf{B}$ be any class closed under conjunction and ψ, ψ' be any members of Δ^\vee , and $x \in \mathcal{V}$ be any variable independent of φ and ψ . Then $x \vee \psi$ and $x \vee \psi'$ are in Δ , and so $(x \vee \psi) \wedge (x \vee \psi') = x \vee (\psi \wedge \psi')$ is in Δ . It follows that $\psi \wedge \psi' \in \Delta^\vee$. ■

4.3 The Approximation Scheme

Recall that for any $\Delta \subseteq \mathbf{B}$ such that Δ is closed under conjunction, we can define

$$\Delta_{\uparrow}(\varphi) = \bigwedge \{ \psi \mid \psi \in \Delta \text{ and } \varphi \models \psi \}$$

and for any $\Delta \subseteq \mathbf{B}$ closed under disjunction, we can define

$$\Delta_{\downarrow}(\varphi) = \bigvee \{ \psi \mid \psi \in \Delta \text{ and } \psi \models \varphi \}$$

Unfortunately, these definitions do not readily lend themselves to practical implementation. However, if we restrict our attention to unbiased decomposable classes, the following equivalent definitions, which are readily implemented, can be used.

Definition 4. For $\Delta \subseteq \mathbf{B}$ and $\varphi \in \mathbf{B}$, we define:

$$\begin{aligned} \mathcal{U}(\varphi) &= \bigwedge_{x \in \mathcal{V}} \left((\Delta_{\uparrow}^{\vee}(\varphi_x^0) \vee x) \wedge (\Delta_{\uparrow}^{\bar{\vee}}(\varphi_x^1) \vee \bar{x}) \right) \wedge \Delta_{\uparrow}^{\mathbf{C}}(\varphi) \\ \mathcal{L}(\varphi) &= \bigvee_{x \in \mathcal{V}} \left((\Delta_{\downarrow}^{\wedge}(\varphi_x^0) \wedge x) \vee (\Delta_{\downarrow}^{\bar{\wedge}}(\varphi_x^1) \wedge \bar{x}) \right) \vee \Delta_{\downarrow}^{\mathbf{C}}(\varphi) \end{aligned}$$

Now we show that, for decomposable closure operators, these definitions indeed specify the Δ envelope and core, respectively.

Theorem 1. For any decomposable class $\Delta \subseteq \mathbf{B}$ such that $1 \in \Delta$ and Δ is closed under conjunction, $\Delta_{\uparrow}(\varphi) = \mathcal{U}(\varphi)$, and for any decomposable class $\Delta \subseteq \mathbf{B}$ closed under disjunction and including 0 , $\Delta_{\downarrow}(\varphi) = \mathcal{L}(\varphi)$.

Proof: We prove only the first part; the second part is its dual. Assume decomposable class $\Delta \subseteq \mathbf{B}$ is closed under conjunction.

$$\Delta_{\uparrow}(\varphi) = \bigwedge \{ \zeta \mid \zeta \in \Delta \text{ and } \varphi \models \zeta \}$$

For every ζ except 0 , we can develop any variable. We handle 0 separately.

$$= \bigwedge_{v \in \mathcal{V}} \bigwedge \left\{ \begin{array}{l} (v \vee \psi) \mid (v \vee \psi) \wedge (\bar{v} \vee \psi') \in \Delta, \psi, \psi' \approx v \\ \wedge (\bar{v} \vee \psi') \mid \text{and } \varphi \models (v \vee \psi) \wedge (\bar{v} \vee \psi') \end{array} \right\} \wedge \Delta_{\uparrow}^{\mathbf{C}}(\varphi)$$

Because Δ is decomposable, we can divide the class membership condition. We can also divide the entailment condition, so we can divide the entire set comprehension into positive and negative halves.

$$= \bigwedge_{v \in \mathcal{V}} \left(\bigwedge \{ v \vee \psi \mid v \vee \psi \in \Delta, \psi \approx v \text{ and } \varphi \models v \vee \psi \} \right) \wedge \Delta_{\uparrow}^{\mathbf{C}}(\varphi)$$

Δ is unbiased and $\psi \approx v$, so $v \vee \psi \in \Delta$ iff $\forall u. u \vee \psi \in \Delta$, and similarly for \bar{v} . Also, $\varphi \models v \vee \psi$ exactly when $\bar{v} \wedge \varphi \models \psi$.

$$= \bigwedge_{v \in \mathcal{V}} \left(\bigwedge \{v \vee \psi \mid \forall u. u \vee \psi \in \Delta, \psi \approx v \text{ and } \bar{v} \wedge \varphi \models \psi\} \right) \wedge \Delta_{\uparrow}^{\mathbf{C}}(\varphi)$$

Since the first set collects $v \vee \psi$, cases of ψ making v false do not matter to the result, and conversely for the second set. For these cases, we need consider only consequences of φ_v^0 (φ_v^1 in the second set). We also observe that the class membership constraint in each set exactly specifies a quotient class. Finally, we factor out the common $v \vee$ or $\bar{v} \vee$ from each set.

$$= \bigwedge_{v \in \mathcal{V}} \left(\bigwedge \left(\begin{array}{l} (v \vee \bigwedge \{\psi \mid \psi \in \Delta^{\vee} \text{ and } \psi \approx v \text{ and } \varphi_v^0 \models \psi\}) \\ (\bar{v} \vee \bigwedge \{\psi \mid \psi \in \Delta^{\bar{\vee}} \text{ and } \psi \approx v \text{ and } \varphi_v^1 \models \psi\}) \end{array} \right) \right) \wedge \Delta_{\uparrow}^{\mathbf{C}}(\varphi)$$

Each set exactly specifies a quotient upper closure operator.

$$\begin{aligned} &= \bigwedge_{v \in \mathcal{V}} \left((v \vee \Delta_{\uparrow}^{\vee}(\varphi_v^0)) \wedge (\bar{v} \vee \Delta_{\uparrow}^{\bar{\vee}}(\varphi_v^1)) \right) \wedge \Delta_{\uparrow}^{\mathbf{C}}(\varphi) \\ &= \mathcal{U}(\varphi) \end{aligned}$$

4.4 Closure Under Instantiation

The algorithms of del Val [8] apply only to classes closed under subsumption. This concept presupposes a clausal representation; from a representation-independent perspective, the equivalent concept is closure under instantiation.

Definition 5. We say a class $\Delta \subseteq \mathbf{B}$ is *closed under instantiation* when for every $\psi \in \Delta$ and $v \in \mathcal{V}$, $\psi_v^0 \in \Delta$ and $\psi_v^1 \in \Delta$. ■

While many important classes, such as **H**, **M**, **K**, and the affine functions are closed under instantiation, some well-known and important classes are not. For example, we can see that both **P** and **D** are not closed under instantiation by observing that $x \rightarrow y$ is both positive and definite, but instantiating y to θ leaves \bar{x} , which is neither positive nor definite.

The characterisations in Section 4.3 do not require closure under instantiation. However, we note an interesting characteristic of classes that do happen to be closed under instantiation.

Proposition 3. For any class $\Delta \subseteq \mathbf{B}$ closed under instantiation, all the quotient classes are subsets of Δ .

Proof: Firstly, $\Delta^{\mathbf{C}} \subseteq \Delta$ by construction. To see that $\Delta^{\vee} \subseteq \Delta$, consider some Δ closed under instantiation, $x \in \mathcal{V}$, and $\psi \in \mathbf{B}$ such that $x \vee \psi \in \Delta$ and $\psi \approx x$; we must show that $\psi \in \Delta$. By closure under instantiation, $(x \vee \psi)_x^0 \in \Delta$. But

$(x \vee \psi)_x^0 = \psi_x^0$, and because $\psi \approx x$, $\psi_x^0 = \psi$. Thus $\psi \in \Delta$. The argument for the other quotient classes is similar. ■

5 Instantiating the Scheme

We can now use the representation-independent proposition from Section 4 to express envelopes for a number of interesting classes. As is easily verified, \mathbf{H} , $\tilde{\mathbf{H}}$, \mathbf{M} , $\tilde{\mathbf{M}}$, \mathbf{K} , \mathbf{L} , \mathbf{S} , \mathbf{P} , \mathbf{D} , \mathbf{C} , and $\mathbf{1}$ are all decomposable and unbiased, and all contain 1 and are closed under conjunction.

5.1 Expressing the Envelopes

First we shall present the quotients of these classes, and then the characterisations of approximation that arise.

Proposition 4.

$$\begin{array}{lll}
 \mathbf{H}^\vee = \tilde{\mathbf{M}} & \mathbf{H}^{\bar{\vee}} = \mathbf{H} & \mathbf{H}^{\mathbf{C}} = \mathbf{C} \\
 \tilde{\mathbf{H}}^\vee = \tilde{\mathbf{H}} & \tilde{\mathbf{H}}^{\bar{\vee}} = \mathbf{M} & \tilde{\mathbf{H}}^{\mathbf{C}} = \mathbf{C} \\
 \mathbf{M}^\vee = \mathbf{M} & \mathbf{M}^{\bar{\vee}} = \mathbf{1} & \mathbf{M}^{\mathbf{C}} = \mathbf{C} \\
 \tilde{\mathbf{M}}^\vee = \mathbf{1} & \tilde{\mathbf{M}}^{\bar{\vee}} = \tilde{\mathbf{M}} & \tilde{\mathbf{M}}^{\mathbf{C}} = \mathbf{C} \\
 \mathbf{K}^\vee = \mathbf{L} & \mathbf{K}^{\bar{\vee}} = \mathbf{L} & \mathbf{K}^{\mathbf{C}} = \mathbf{C} \\
 \mathbf{L}^\vee = \mathbf{C} & \mathbf{L}^{\bar{\vee}} = \mathbf{C} & \mathbf{L}^{\mathbf{C}} = \mathbf{C} \\
 \mathbf{S}^\vee = \tilde{\mathbf{V}} & \mathbf{S}^{\bar{\vee}} = \mathbf{V} & \mathbf{S}^{\mathbf{C}} = \mathbf{C} \\
 \mathbf{V}^\vee = \mathbf{C} & \mathbf{V}^{\bar{\vee}} = \mathbf{1} & \mathbf{V}^{\mathbf{C}} = \mathbf{C} \\
 \tilde{\mathbf{V}}^\vee = \mathbf{1} & \tilde{\mathbf{V}}^{\bar{\vee}} = \mathbf{C} & \tilde{\mathbf{V}}^{\mathbf{C}} = \mathbf{C} \\
 \mathbf{P}^\vee = \mathbf{B} & \mathbf{P}^{\bar{\vee}} = \mathbf{P} & \mathbf{P}^{\mathbf{C}} = \mathbf{1} \\
 \mathbf{D}^\vee = \tilde{\mathbf{M}} & \mathbf{D}^{\bar{\vee}} = \mathbf{D} & \mathbf{D}^{\mathbf{C}} = \mathbf{1}
 \end{array}$$

Proof: All cases follow easily from the well-known syntactic characterisations of the classes and the definitions of the quotient classes. ■

Now we are ready to show the instantiations of the general characterisation to the individual classes. Note that where a quotient class is $\mathbf{1}$, it can be trivially omitted, as it always returns 1 . Similarly, where a quotient class is \mathbf{B} , it need not be applied to its argument as it is the identity operator. Also note that the conjunct $\mathbf{C}_\uparrow(\varphi)$ has no effect if φ is satisfiable, and for unsatisfiable φ , it becomes \emptyset .

Corollary 1. Let φ be a Boolean function. Then

$$\begin{aligned}
\mathbf{H}_\uparrow(\varphi) &= \bigwedge_{v \in \mathcal{V}} ((\widetilde{\mathbf{M}}_\uparrow(\varphi_v^0) \vee v) \wedge (\mathbf{H}_\uparrow(\varphi_v^1) \vee \bar{v})) \wedge \mathbf{C}_\uparrow(\varphi) \\
\widetilde{\mathbf{H}}_\uparrow(\varphi) &= \bigwedge_{v \in \mathcal{V}} ((\widetilde{\mathbf{H}}_\uparrow(\varphi_v^0) \vee v) \wedge (\mathbf{M}_\uparrow(\varphi_v^1) \vee \bar{v})) \wedge \mathbf{C}_\uparrow(\varphi) \\
\mathbf{M}_\uparrow(\varphi) &= \bigwedge_{v \in \mathcal{V}} ((\mathbf{M}_\uparrow(\varphi_v^0) \vee v) \wedge (\mathbf{1}_\uparrow(\varphi_v^1) \vee \bar{v})) \wedge \mathbf{C}_\uparrow(\varphi) \\
\widetilde{\mathbf{M}}_\uparrow(\varphi) &= \bigwedge_{v \in \mathcal{V}} ((\mathbf{1}_\uparrow(\varphi_v^0) \vee v) \wedge (\widetilde{\mathbf{M}}_\uparrow(\varphi_v^1) \vee \bar{v})) \wedge \mathbf{C}_\uparrow(\varphi) \\
\mathbf{K}_\uparrow(\varphi) &= \bigwedge_{v \in \mathcal{V}} ((\mathbf{L}_\uparrow(\varphi_v^0) \vee v) \wedge (\mathbf{L}_\uparrow(\varphi_v^1) \vee \bar{v})) \wedge \mathbf{C}_\uparrow(\varphi) \\
\mathbf{L}_\uparrow(\varphi) &= \bigwedge_{v \in \mathcal{V}} ((\mathbf{C}_\uparrow(\varphi_v^0) \vee v) \wedge (\mathbf{C}_\uparrow(\varphi_v^1) \vee \bar{v})) \wedge \mathbf{C}_\uparrow(\varphi) \\
\mathbf{S}_\uparrow(\varphi) &= \bigwedge_{v \in \mathcal{V}} ((\widetilde{\mathbf{V}}_\uparrow(\varphi_v^0) \vee v) \wedge (\mathbf{V}_\uparrow(\varphi_v^1) \vee \bar{v})) \wedge \mathbf{C}_\uparrow(\varphi) \\
\mathbf{V}_\uparrow(\varphi) &= \bigwedge_{v \in \mathcal{V}} (\mathbf{C}_\uparrow(\varphi_v^0) \vee v) \wedge \mathbf{C}_\uparrow(\varphi) \\
\widetilde{\mathbf{V}}_\uparrow(\varphi) &= \bigwedge_{v \in \mathcal{V}} (\mathbf{C}_\uparrow(\varphi_v^1) \vee \bar{v}) \wedge \mathbf{C}_\uparrow(\varphi) \\
\mathbf{P}_\uparrow(\varphi) &= \bigwedge_{v \in \mathcal{V}} ((\varphi_v^0 \vee v) \wedge (\mathbf{P}_\uparrow(\varphi_v^1) \vee \bar{v})) \\
\mathbf{D}_\uparrow(\varphi) &= \bigwedge_{v \in \mathcal{V}} ((\widetilde{\mathbf{M}}_\uparrow(\varphi_v^0) \vee v) \wedge (\mathbf{D}_\uparrow(\varphi_v^1) \vee \bar{v})) \quad \blacksquare
\end{aligned}$$

Other instances exist, most notably we can generalise \mathbf{L} (1-CNF) and \mathbf{K} (2-CNF) to k -CNF: $(k+1)$ -CNF $^\vee = (k+1)$ -CNF $^{\bar{\vee}} = k$ -CNF, and $(k+1)$ -CNF $^{\mathbf{C}} = \mathbf{C}$. We can similarly extend these results to k -quasi-Horn.

5.2 Algorithmic Aspects

Corollary [1](#) characterises the envelopes for a number of interesting function classes in a representation-independent manner. They do not always suggest the most efficient way of calculating envelopes, which in general depends on how Boolean functions are represented. We also note that there are cases where an envelope cannot be provided in the absence of information about the “variables of interest”. For example, we cannot say what the \mathbf{D} envelope of \bar{x} is, unless we know the set of variables of which \bar{x} is supposed to be a function. Using Church’s lambda notation helps; using it we can state for example that $\mathbf{D}_\uparrow(\lambda x, y. \bar{x}) = x \rightarrow y$ whereas $\mathbf{D}_\uparrow(\lambda x, y, z. \bar{x}) = x \rightarrow (y \wedge z)$.

It is interesting to compare our characterisations, which were derived using Boolean development, with recursive definitions used for ROBDDs (also resting

on Boolean development). We present below the algorithms for \mathbf{H} and $\widetilde{\mathbf{M}}$ —algorithms for the other classes can be derived [20].

Recall that binary decision diagrams are defined inductively:

- 0 is a BDD.
- 1 is a BDD.
- If $x \in \mathcal{V}$ and R_1 and R_2 are BDDs then $\text{ite}(x, R_1, R_2)$ is a BDD.

and the meaning of a BDD is given as follows.

$$\begin{aligned} \llbracket 0 \rrbracket &= 0 \\ \llbracket 1 \rrbracket &= 1 \\ \llbracket \text{ite}(x, R_1, R_2) \rrbracket &= (x \wedge \llbracket R_1 \rrbracket) \vee (\bar{x} \wedge \llbracket R_2 \rrbracket) \end{aligned}$$

ROBDDs are then BDDs with a fixed variable order, satisfying the constraints that in any BDD $\text{ite}(x, R_1, R_2)$, $R_1 \neq R_2$, and that for any distinct BDDs R_1 and R_2 appearing in R , $\llbracket R_1 \rrbracket \neq \llbracket R_2 \rrbracket$. As is common, we use a function $\text{mknd}(x, R_1, R_2)$ to create all ROBDD nodes according to these rules:

1. If $R_1 = R_2$, return R_1 instead of a new node, as $\llbracket \text{ite}(x, R_1, R_2) \rrbracket = \llbracket R_1 \rrbracket$.
2. If an identical ROBDD was previously built, return that one instead of a new one; this is accomplished by keeping a hash table, called the *unique table*, of all previously created nodes [2].
3. Otherwise, return $\text{ite}(x, R_1, R_2)$.

Algorithm 1. To find the Horn envelope of an ROBDD:

$$\begin{array}{ll} \mathbf{H}_\uparrow(0) = 0 & \widetilde{\mathbf{M}}_\uparrow(0) = 0 \\ \mathbf{H}_\uparrow(1) = 1 & \widetilde{\mathbf{M}}_\uparrow(1) = 1 \\ \mathbf{H}_\uparrow(\text{ite}(x, R_1, R_2)) & \widetilde{\mathbf{M}}_\uparrow(\text{ite}(x, R_1, R_2)) \\ \quad = \text{mknd}(x, R^t, R^f) & \quad = \text{mknd}(x, R'_1, \text{or}(R'_1, R'_2)) \\ \quad \textbf{where } R' = \mathbf{H}_\uparrow(\text{or}(R_1, R_2)) & \quad \textbf{where } R'_1 = \widetilde{\mathbf{M}}_\uparrow(R_1) \\ \quad \textbf{and } R^t = \mathbf{H}_\uparrow(R_1) & \quad \textbf{and } R'_2 = \widetilde{\mathbf{M}}_\uparrow(R_2) \\ \quad \textbf{and } R^f = \text{and}(\widetilde{\mathbf{M}}_\uparrow(R_2), R') & \end{array}$$

■

Note that $\widetilde{\mathbf{M}}_\uparrow(\varphi \vee \psi) = \widetilde{\mathbf{M}}_\uparrow(\varphi) \vee \widetilde{\mathbf{M}}_\uparrow(\psi)$.

6 Discussion

Several contributors to the field of approximate knowledge compilation have suggested departures from the classical setting, regarding both the classes of Boolean functions used, and the data structures used to represent these functions. The lattice-theoretic concepts of upper and lower closure operators provide an abstract and useful lens for the study of envelopes and cores in propositional logic, independent of representation. In the first half of this paper we have put forward this view in greater detail. The framework is general. While we focus

on lattices of Boolean functions, note that no assumptions were made about the properties of the lattices. In particular they need not be Boolean lattices, that is, they are neither restricted to be complemented nor distributive. Indeed, the majority of the function classes considered do not form complemented lattices, and many are not distributive. For example, to see that \mathbf{H} is not distributive, note that x , y , and $x \leftrightarrow y$ are all Horn, but

$$(x \sqcup y) \sqcap (x \leftrightarrow y) = 1 \wedge (x \leftrightarrow y) \neq x \wedge y = (x \sqcap (x \leftrightarrow y)) \sqcup (y \sqcap (x \leftrightarrow y))$$

where \sqcap is the meet operation on \mathbf{H} (that is, conjunction), and \sqcup is the join (which is not disjunction).

Our main contribution, expressed as Theorem 11, is a generic characterisation of envelopes and cores in a large variety of Boolean function classes. Many instantiations of the theorem, including versions for Horn and Krom functions, are provided in Section 5.

It remains to be seen to what extent the algorithms we have derived can be made efficient for various representations. So far we are in the process of implementing a range of the algorithms for ROBDDs, together with algorithms for finding least upper bounds and greatest lower bounds for sets of functions in various classes. (The use of the term “LUB” in much of the literature on Horn approximation is somewhat incongruous with standard usage, and “GLB” even more so.) Another challenge is to develop an algorithm to produce affine envelopes of ROBDDs.

We would also like to better understand the relations between the framework offered by del Val 8 and the one proposed here. For example, at least on the surface it would seem that closure under subsumption corresponds exactly to closure under instantiation (by the latter we mean $\varphi_x^0, \varphi_x^1 \in \Delta$ whenever $\varphi \in \Delta$, for all $x \in \mathcal{V}$). However, we note that our development of the generic algorithm did not require an assumption about closure under instantiation.

References

1. Birkhoff, G.: Lattice Theory. American Mathematical Society, (3rd edn.) (1973)
2. Brace, K., Rudell, R., Bryant, R.: Efficient implementation of a BDD package. In: Proc. Twenty-seventh ACM/IEEE Design Automation Conf., pp. 40–45 (1990)
3. Cadoli, M., Scarcello, F.: Semantical and computational aspects of Horn approximations. *Artificial Intelligence* 119, 1–17 (2000)
4. Codish, M., Søndergaard, H., Stuckey, P.J.: Sharing and groundness dependencies in logic programs. *ACM Transactions on Programming Languages and Systems* 21(5), 948–976 (1999)
5. Cousot, P., Cousot, R.: Static determination of dynamic properties of recursive procedures. In: Neuhold, E.J. (ed.) *Formal Description of Programming Concepts*, pp. 237–277. North-Holland (1978)
6. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: Proc. Sixth ACM Symp. Principles of Programming Languages, pp. 269–282. ACM Press, New York (1979)
7. Dechter, R., Pearl, J.: Structure identification in relational data. *Artificial Intelligence* 58, 237–270 (1992)

8. del Val, A.: First order LUB approximations: Characterization and algorithms. *Artificial Intelligence* 162, 7–48 (2005)
9. Giacobazzi, R.: *Semantic Aspects of Logic Program Analysis*. PhD thesis, University of Pisa, Italy (1993)
10. Halmos, P.R.: *Lectures on Boolean Algebras*. Springer, Heidelberg (1963)
11. Horiyama, T., Ibaraki, T.: Ordered binary decision diagrams as knowledge-bases. *Artificial Intelligence* 136, 189–213 (2002)
12. Horiyama, T., Ibaraki, T.: Translation among CNFs, characteristic models and ordered binary decision diagrams. *Inf. Processing Letters* 85, 191–198 (2003)
13. Kautz, H., Kearns, M., Selman, B.: Horn approximations of empirical data. *Artificial Intelligence* 74, 129–145 (1995)
14. Kavvadias, D., Papadimitriou, C., Sideri, M.: On Horn envelopes and hypergraph transversals. In: Ng, K.W., Balasubramanian, N.V., Raghavan, P., Chin, F.Y.L. (eds.) *ISAAC 1993*. LNCS, vol. 762, pp. 399–405. Springer, Heidelberg (1993)
15. Khardon, R.: Translating between Horn representations and their characteristic models. *Journal of Artificial Intelligence Research* 3, 349–372 (1995)
16. Ore, O.: Combinations of closure relations. *Ann. Math.* 44(3), 514–533 (1943)
17. Pelletier, F.J., Martin, N.M.: Post’s functional completeness theorem. *Notre Dame Journal of Formal Logic* 31(2) (1990)
18. Post, E.L.: *The Two-Valued Iterative Systems of Mathematical Logic*. Princeton University Press, 1941. Reprinted in Davis, M., *Solvability, Provability, Definability: The Collected Works of Emil L. Post*, pp. 249–374, Birkhäuser (1994)
19. Rudeanu, S.: *Boolean Functions and Equations*. North-Holland (1974)
20. Schachte, P., Søndergaard, H.: Closure operators for ROBDDs. In: Emerson, E.A., Namjoshi, K. (eds.) *VMCAI 2006*. LNCS, vol. 3855, pp. 1–16. Springer, Heidelberg (2005)
21. Schaefer, T.J.: The complexity of satisfiability problems. In: *Proc. Tenth Ann. ACM Symp. Theory of Computing*, pp. 216–226 (1978)
22. Selman, B., Kautz, H.: Knowledge compilation and theory approximation. *Journal of the ACM* 43(2), 193–224 (1996)
23. Ward, M.: The closure operators of a lattice. *Ann. Math.* 43(2), 191–196 (1942)
24. Zanuttini, B.: Approximating propositional knowledge with affine formulas. In: (ECAI’02). *Proceedings of the Fifteenth European Conference on Artificial Intelligence*, pp. 287–291. IOS Press, Amsterdam (2002)
25. Zanuttini, B.: Approximation of relations by propositional formulas: Complexity and semantics. In: Koenig, S., Holte, R.C. (eds.) *SARA 2002*. LNCS (LNAI), vol. 2371, pp. 242–255. Springer, Heidelberg (2002)

An Analysis of Map-Based Abstraction and Refinement

Nathan Sturtevant and Renee Jansen

Department of Computing Science, University of Alberta, Edmonton, Alberta, T6G 2E8, Canada
{nathanst, maaike}@cs.ualberta.ca

Abstract. A variety of techniques have been introduced over the last decade for abstracting search graphs and then using these abstractions for search. While some basic work has been done to predict the value of an abstraction mechanism, the results have not been validated in practice. In this paper we analyze a variety of old and new abstraction mechanisms in a pathfinding testbed and show that the work done in abstraction-based refinement-style search can be predicted by the diameter and size of abstract nodes.

1 Introduction

Search is a widely studied task in artificial intelligence, due to the fact that many problems such as pathfinding and scheduling can be solved using search techniques. The traditional and widely used search algorithm is A^* [1], which uses a heuristic function to guide its search. Unfortunately, this algorithm is computationally expensive: the amount of work required can be exponential in the length of the solution.

One way to deal with this problem is by abstracting the search space. In particular, it is possible to create a simpler search graph by representing a number of nodes of the original search graph by a single node in an abstract search graph. Abstract edges are then added based on the edges that exist in the original search graph. This can be done recursively, giving a hierarchy of abstractions. An approximate solution can be found by a search in the abstract graph, which can then be *refined* to a solution in the original search space.

Abstraction methods have been studied in a number of places. For example, Holte *et al.* proposed the STAR abstraction, in which a node and all nodes within some predefined radius are abstracted together [2]. Botea *et al.* developed an abstraction method specifically designed for gridworlds, which divides the grid into square clusters [3]. Another abstraction method, devised by Sturtevant and Buro, takes cliques in the original search graph and abstracts them into a single node in the abstract graph [4]. Abstraction has been used in other domains such as robotics [5] and planning [6].

Holte *et al.*'s STAR abstraction was developed as a result of a theoretical analysis of the amount of work required to find a solution to a search problem using abstractions of the search space [2]. The analysis shows that, in order to minimize the total amount of work done, it is desirable to minimize the maximum length of a path between any two nodes inside an abstract node, while maximizing the number of nodes that are combined into a single abstract node.

The goal of this paper is to verify the analysis done by Holte *et al.* of the amount of work done during a search which uses search space abstractions. We will first compare

the amount of work done with a variety of both new and old abstraction methods, and then show that the analysis of Holte *et al.* holds in practice.

1.1 Problem Definition

A search problem can be formally defined as the tuple (S, A, c, s_0, S_g) , where S denotes the set of states in the environment, A denotes the set of actions, $c(s, a)$ is the cost of taking action $a \in A$ in state $s \in S$, $s_0 \in S$ is the start state, and $S_g \subseteq S$ is the set of goal states. The search space can be represented as a graph $G = (V, E)$, where V is the set of vertices (nodes), representing the set of states S , and E is the set of edges, representing the actions A . The weight of an edge is defined as the cost of performing action a in state s . In this paper, we will assume that the edges of the search graph are undirected, implying that the cost of going from state s to state s' is equal to the cost of going from state s' to s .

We look specifically at problems from the pathfinding domain. That is, underlying the graph representation of the world there is a grid-based map. Cardinal moves on the map (N, S, E, W) have cost 1 while diagonal moves have cost $\sqrt{2}$. We choose this focus, because pathfinding is currently a widely studied and well-motivated area, with applications in areas including robotics and computer games.

2 Abstraction Mechanisms

We first define an abstraction formally, and then present 5 different abstraction mechanisms which we will use experimentally to test the predictions made by Holte *et al.* [2]. The radius and clique abstractions have been described elsewhere; the other abstraction mechanisms are presented for the first time here.

2.1 Automatic State Abstraction

Formally, an abstraction is a graph homomorphism ϕ from a graph G_1 to a graph G_2 which maps nodes from G_1 to nodes in G_2 . An edge e is added between two abstract nodes s_1 and s_2 whenever there is at least one edge e' between s'_1 and s'_2 such that $\phi(s'_1) = s_1$ and $\phi(s'_2) = s_2$. We can abstract the search graph recursively, giving an abstraction hierarchy of α levels, where level 0 is the original search graph and level α is the topmost abstract level. We will introduce a variety of such homomorphisms.

While edge costs are well-defined in the original problem space, we define the location of a node s in abstract space as the average location of all the nodes abstracted by s . This means that we can use the location of abstract nodes as a heuristic for searching the abstract graph. This heuristic will be admissible in abstract space, but not in the original problem space. In domains other than pathfinding, edges can have uniform cost.

2.2 Clique Abstraction

The clique abstraction (CA) was initially introduced by Sturtevant and Buro [4]. The idea behind this abstraction, as the name suggests, is to abstract cliques in each level of

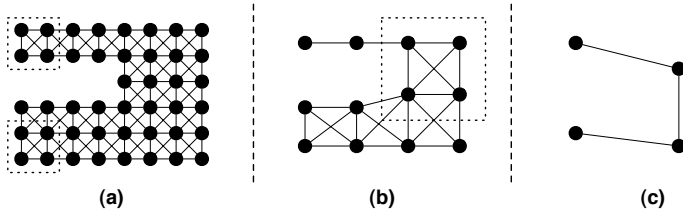


Fig. 1. Clique abstraction

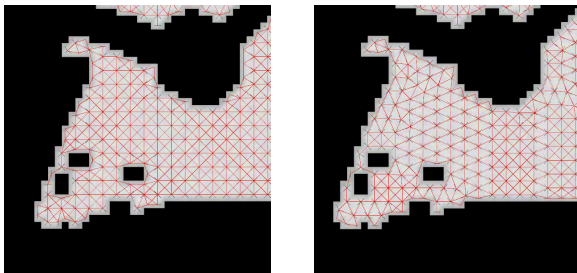


Fig. 2. Uniform (left) and non-uniform (right) abstractions

the abstraction. Every node in a clique will be no more than one edge away from every node, which is a desirable property, according to the theoretical evaluation discussed in the next section. In a two-dimensional, octile-connected map, the maximum clique size is 4 nodes, which makes the clique-finding problem tractable.

We illustrate the clique-abstraction process in Figure 1. The initial graph is shown in the portion of this figure labeled (a). Two sample cliques are indicated with a dotted line. The middle figure, (b), shows one possible way the first graph can be abstracted. Note that where four-cliques cannot be abstracted, smaller cliques are removed instead. This abstraction mechanism can be applied once more to (b) to obtain the graph in (c). In (b), the marked clique is removed first, followed by the only other four-clique. The remaining pairs of nodes will be abstracted together. Depending on the order in which nodes are considered and the policy for abstracting nodes with only a single neighbor, the final graph will either take one or two steps to abstract until it is represented by just a single node.

In this paper we will use two different approaches for building a clique abstraction. The first method, $CA(n)$ [normalized], builds the abstract graph in a more uniform manner by relying on knowledge of the underlying map. $CA(n)$ first abstracts 4-cliques in a uniform manner across open areas of the map before considering the rest of the map. The second implementation of the clique abstraction, $CA(i)$ [irregular], does not rely on knowledge of an underlying map and thus builds less uniform abstractions. We demonstrate the difference between a uniform and non-uniform abstraction in Figure 2. The lines in this figure represent edges, with nodes implicit at the ends of edges. In the left portion of the figure, most of the map has been abstracted in uniform squares. Only

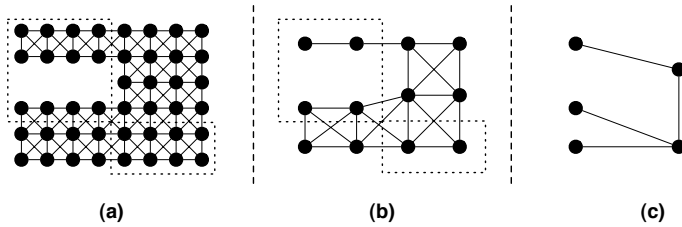


Fig. 3. Sector abstraction

the edges of the map are less uniform. Because the underlying topology is not known in the right portion of this figure, the abstraction is much less uniform.

There will be some cases where the abstraction procedure fails to find a clique among the remaining unabstracted nodes. In this case, these nodes can be passed through to the next abstraction level instead of being abstracted with their neighbors. In the case of nodes with only a single neighbor, we choose to abstract them into their neighbor regardless of whether they form a non-trivial clique or not.

2.3 Sector Abstraction

The sector abstraction (SA) is inspired by the abstraction used by HPA* [3], and is limited to grid-based maps. The sector abstraction is parameterized by a fixed sector size, k . At the first level of abstraction, sectors of size $k \times k$ are overlaid onto the map. Within each sector, a breadth-first search is used to determine connected components, each of which becomes an abstract node. At the i th level of abstraction, sectors of size $k^i \times k^i$ are used. Note that with an empty map and a sector size of 2, the clique abstraction and sector abstraction will be identical.

We demonstrate this abstraction mechanism with a sector size of 2 in Figure 3. In this example, the clique abstraction and sector abstraction both abstract the initial graph in the same manner. In graphs (a) and (b), two of the sectors (4×4) used for the building graph (c) pass are marked by dotted lines. Because only nodes which form a connected component within a single sector can be abstracted together, the top left sector becomes two separate nodes when abstracted. This results in one extra node in the most abstract graph on the right, (c). If we were to apply one more level of abstraction, the entire graph would be immediately abstracted into a single node, because all nodes in this graph are connected within an 8×8 sector.

2.4 Radius Abstraction

Holte *et al.* suggested an abstraction mechanism they called the STAR abstraction [2]. We use what is essentially the same mechanism, but refer to it as the radius abstraction (RA), which we feel is a more evocative description. The radius abstraction works by first selecting an unabstracted node. All neighboring nodes within a fixed radius, r , of this node are then abstracted together into the same abstract node. The radius, r , is the depth limit (in edges) on a breadth-first search which finds the neighbors to abstract. The

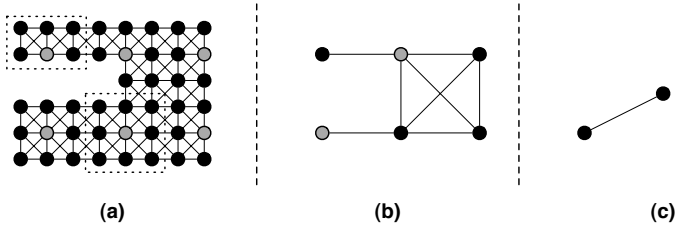


Fig. 4. Radius one abstraction

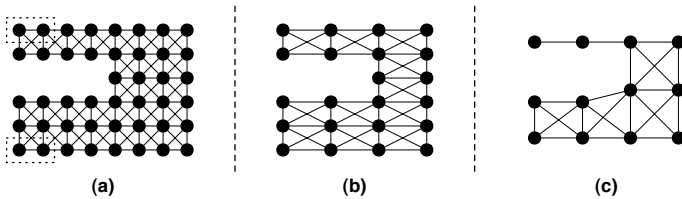


Fig. 5. Line abstraction

radius abstraction procedure is simple and can be applied to any graph. We demonstrate the radius abstraction in Figure 4.

Our implementation of the radius abstraction chooses the next node to abstract at random; however, in this example we choose the nodes to be abstracted quite carefully. In the left part of Figure 4 (a), we mark in gray the nodes which are selected for abstraction. The immediate neighbors ($r = 1$) of these nodes are then abstracted together. In the first abstract graph, (b), there are only 6 nodes, and again, we mark in gray the nodes which are selected to drive the abstraction process. The resulting graph, (c), has two nodes, and will be fully abstracted at the next level of abstraction. The radius abstraction will remove more nodes in each step than the clique abstraction. A radius 1 abstraction can be quite similar to a sector abstraction with $k = 3$.

2.5 Line Abstraction

The line abstraction (LA) finds sequences of nodes length k , and abstracts them together. In this paper we experiment with two variants of the line abstraction. One variant abstracts the graph uniformly, as we will do in the example below. The other variant just selects nodes to abstract at random and is much less uniform. We vary the maximum length of the abstracted line between 2 and 6.

We demonstrate the line abstraction with $k = 2$ in Figure 5. First, we attempt to abstract each node with its neighbor to the right. This takes the original graph, (a) and transforms it into the graph (b). In the next step we attempt to abstract each node with its neighbor below. This results in the graph on the far right, (c), which is identical to the graph produced by clique-abstraction in a single step. When done uniformly, the line abstraction proceeds in this manner, first abstracting horizontally and then vertically.

2.6 Node-Limit Abstraction

The node limit abstraction (NLA) has a single parameter k , the number of nodes to abstract together. Given an initial node, a breadth-first search is performed until k unabstracted nodes have been visited. These nodes are then abstracted into a single abstract node. The node-limit abstraction and line abstraction are the same when $k = 2$. If k is defined dynamically as the number of neighbors within a radius r , the node-limit abstraction will be the same as the radius abstraction. In this paper we use a fixed k for all nodes.

3 Abstraction Analysis

In this section we analyze the complexity of using an abstraction hierarchy to find a path through a search graph. For the moment we will consider using an algorithm which first finds a path at the highest possible level of abstraction, and then successively refines this path until a path is found in the original graph. This analysis is originally due to Holte *et al.* [2] and we follow their derivation closely here.

During the refinement process, a node s at some abstract level i is replaced by a series of nodes in level $i - 1$. This is done by finding a path p in level $i - 1$ consisting of nodes which are mapped to s . In particular, if we let the neighbours of s along the path at level i be t and u , the first and last nodes on path p must have neighbours t' and u' such that $\phi(t') = t$ and $\phi(u') = u$. We say refinement is *monotonic* if no backtracking needs to be done across levels. This will be the case throughout this paper.

The total amount of work done in finding a solution consists of the refinement costs at each level. If we assume that every abstract graph is strictly smaller than the graph it abstracts, the solution in the abstract graph at level α is trivial since this graph will only have a single node (one node per connected component in the original graph). In each refinement step, every node in the solution at level i is replaced by a sequence of nodes at level $i - 1$. If we let the length of the path at level i be denoted by λ_i , and the work required to replace a node at level i by a sequence of nodes at level $i - 1$ by ω , then the total work done in refining from level i to level $i - 1$ is $\omega\lambda_i$. The *expansion factor* χ is defined to be λ_i/λ_{i-1} , giving $\lambda_i = \chi\lambda_{i-1} = \chi^{\alpha-i}$.

If we let ω and χ be the worst cases, this gives a bound on the total work done to find a solution:

$$Total\ Work \leq \omega \sum_{i=1}^{\alpha} \chi^{\alpha-i}$$

This is equivalent to:

$$Total\ Work \leq \omega \sum_{i=0}^{\alpha-1} \chi^i$$

Furthermore, we know that χ is upper-bounded by d , the maximum diameter of any abstract state. The *diameter* is the maximum distance between any two states in that abstract state. (This distance is defined as the number of *nodes* on a path.) This gives:

$$Total\ Work \leq \omega \sum_{i=0}^{\alpha-1} d^i$$

The sum over d^i can be represented by the closed form formula $(d^\alpha - 1)/(d - 1)$. If the diameter is at least 2, we can replace this with d^α . We will assume that $n \geq 2$ and n is the same for all states. After replacing α with $\log_n N$, where n is the number of states mapped to a single abstract state and N is the number of states in the original space, the bound on the total work can then be expressed as:

$$Total\ Work \leq \omega n^{(\ln N)/(\ln d)}$$

This function is symmetric in n and N , so we can swap them, giving:

$$Total\ Work \leq \omega N^{(\ln n)/(\ln d)}$$

This shows that if we ignore ω , the total work can be minimized by minimizing d and maximizing n , *i.e.*, by making the maximum distance between any two nodes in an abstract state as small as possible and making the number of states which map to an abstract state as large as possible.

Consider the case where we choose the d as small as possible, *i.e.*, $d = 2$. In this case there is an edge between any pair of nodes that make up the abstract node: the nodes form a clique. Based on the above analysis we need to maximize the number of nodes that are abstracted together; we want cliques that are as large as possible.

Whereas Holte *et al.* were interested in generating an upper bound on the amount of work required to do refinement, we are also interested in finding a measure that will be predictive of the actual work performed. We note that measuring the maximum diameter of the abstraction, d , can be a poor estimate of the cost required to refine a path through a node in practice. For abstractions that are relatively symmetric, such as the clique abstraction, d is likely a good estimate of the cost of refinement. On the other hand, the line abstraction will have $d \approx k$, but there are many short paths through an abstract node, so the average path length through an abstract node will be shorter than d . (This can be verified by Table 1 later in the paper.)

We therefore propose a different measure of d , which we will call d_E . Instead of measuring the maximum distance between states abstracted within a node, we measure the expected cost given that we enter the node from a random edge e_1 and exit from a different edge e_2 . For instance, in Figure 6 there are 11 edges at level i by which we can enter or exit the node when doing refinement from level $i + 1$ to level i . To enter the edge marked e_1 and exit the edge marked e_2 we would have to traverse a single internal edge. We also add the cost of entering on edge e_1 , so this path has cost 2. (Assuming uniform edge costs.) The expected cost to refine the abstract node given that we enter on edge e_1 is $\frac{1 \times 2 + 8 \times 2}{10} = 1.8$. To compute d_E in this example we would perform the same computation for all edges at level i which are external to the abstract node at level $i + 1$.

Finally, we note that this discussion has ignored the ω term, which is the work required to refine a single node. This is directly related to n , so that although we want to increase n to minimize the total work, this has a secondary effect which increases work, so increasing n may not be as effective as decreasing d .

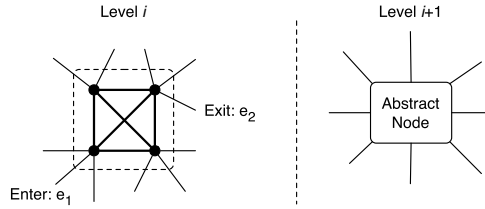


Fig. 6. Measuring the expected diameter of an abstract node

4 Experimental Results

One goal of this paper is to experimentally analyze the previous theoretical results. We begin by measuring properties from the previous section for the different abstractions over 116 maps extracted from Baldur's Gate II (a role-playing game) and Warcraft III (a real-time strategy game). Two examples of these maps can be found in Figure 7. For our experiments we scaled the maps to 512×512 so they are all similarly sized. In Table 1 we list the different abstraction types and the parameters used for each. We measured each of the theoretical properties averaged over all nodes at all levels of the maps.

The values which describe for each abstraction type are not surprising. For instance, the line abstraction ($k = 2$) abstracts, on average, just under two nodes at a time. The

Table 1. The abstraction properties for each abstraction type: average nodes abstracted, the average of the max diameter and the average of the expected diameter of an abstract node

Abstraction Type	Abbrev.	Avg. Nodes (n)	Avg. Diameter (d)	Expected Diameter (d_E)
Clique (Non-uniform)	CA(i)	3.60	0.96	1.72
Clique (Uniform)	CA(n)	3.71	0.97	1.73
Sector 2	SA(2)	3.70	0.98	1.74
Sector 3	SA(3)	7.51	1.89	2.29
Radius 1	RA(1)	8.15	2.36	2.31
Radius 2	RA(2)	10.17	2.80	2.53
NodeLimited 3	NLA(3)	2.57	1.00	1.61
NodeLimited 5	NLA(5)	3.96	1.65	1.86
NodeLimited 6	NLA(6)	4.78	1.84	2.00
Line 2 (Non-uniform)	LA(2, i)	1.86	0.86	1.46
Line 3 (Non-uniform)	LA(3, i)	2.59	1.27	1.67
Line 4 (Non-uniform)	LA(4, i)	3.18	1.65	1.80
Line 5 (Non-uniform)	LA(5, i)	3.58	1.82	1.87
Line 6 (Non-uniform)	LA(6, i)	3.78	1.87	1.87
Line 2 (Uniform)	LA(2, n)	1.97	0.97	1.52
Line 3 (Uniform)	LA(3, n)	2.89	1.86	1.90
Line 4 (Uniform)	LA(4, n)	3.78	2.72	2.22
Line 5 (Uniform)	LA(5, n)	4.50	3.35	2.45
Line 6 (Uniform)	LA(6, n)	5.20	3.97	2.66

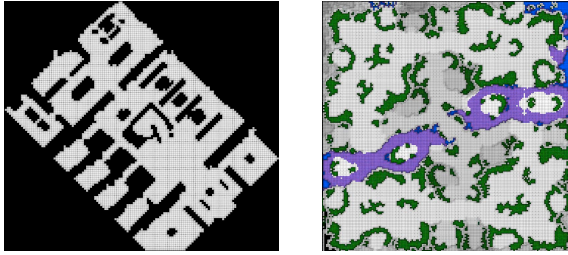


Fig. 7. Two of the maps used in the experiments

uniform abstractions abstract more nodes than the non-uniform abstractions, as seen in Figure 2. The third column in Table 1. Avg. Diameter, is the maximum length of the shortest path between any two states abstracted into a node. For the clique, line ($k = 2$), and sector ($k = 2$) abstractions this value can be at most 1, but is 0 in the cases where a single node cannot be abstracted with its neighbors, resulting in an average value just below 1. We might expect the abstraction with the most nodes, RA(2), to have a diameter of 4, twice the radius. However, since this abstraction is not performed in a uniform manner, its average is lower than the maximum possible.

The final column is the expected number of edges that would be traversed given that we randomly select an incoming and outgoing edge and then measure the number of edges needed to traverse through the node, including the incoming edge. Because we include the incoming edge, we expect this value to be at most 1 larger than the maximum diameter. This value is more indicative of the cost of traveling through a node than the maximum diameter of a node.

Consider, for instance, two nodes abstracted by the line abstraction ($k = 2$). If they both have the same number of edges, we expect that a random path will pass through a single node half the time (cost 1), and pass through both nodes half the time (cost 2). Thus, the expected diameter would be 1.5. The expected diameter is slightly higher for the LA(2, n) abstraction because we do not consider the possibility of entering and exiting from the same edge, as this can never be part of a refined path. So there is a slightly higher chance of traveling through a node than entering and exiting it directly.

4.1 Abstraction-Based Search Algorithms

In this paper, we use the PRA* algorithm [4] to find paths in each of the abstraction hierarchies. PRA* stands for Partial-Refinement A*, although we use the PRA*(∞) variant, which does full refinement of paths. Given start and goal nodes, PRA* successively maps these nodes into the next higher abstract graph until the abstract start and goal are connected by a single edge. If this occurs at level ℓ in the abstraction hierarchy, PRA* then uses A* at level $\ell/2$ to find a path between the abstract start and goal nodes. A* is used to find paths at successively lower levels of abstraction as well, except that it is constrained to a corridor. The corridor at level ℓ includes all nodes which abstract into either the solution at level $\ell + 1$ or which abstract into a node which neighbors the solution at level $\ell + 1$. PRA* returns paths that are very close to optimal both because

it begins planning at the middle of the abstraction hierarchy, and because it widens the corridor used for search.

As a comparison, we use a simple refinement algorithm which follows the theoretical derivation from the last section more closely. This simple refinement algorithm is like PRA* except that it starts at the top of the abstraction hierarchy, and does not expand the corridor during the refinement portion of the search. This generally decreases the work that must be done, but results in lower-quality (*i.e.*, longer) paths.

4.2 Search Costs

Given that we have measured the properties of our abstraction, we are interested in measuring the cost of pathfinding using these abstractions and comparing them to see if there is a correlation between the number of abstracted nodes, n , and the diameter of the abstraction, d or d_E .

Our experiments were conducted as follows. We used the same 116 maps as above. On each map, we chose random paths from length 1 to length 512, and placed 10 paths in each of 128 buckets. The first bucket has paths length $(0, 4]$, while the 128th bucket has paths length $(508, 512]$. In total, we have 114,131 paths over all the maps. We then computed a path between these nodes using PRA* and the simple refinement variant of PRA* previously described. We did this for each of the abstractions in Table 1.

Table 2. A comparison of suboptimality with PRA*(∞) and simple refinement

	PRA*(∞)		Simple Refinement	
	suboptimality	nodes	suboptimality	nodes
CA(n)	0.04	5301	12.04	1840
CA(i)	0.19	5228	15.81	2062
SA(2)	0.04	5416	6.85	1716
SA(3)	0.02	6232	6.19	1826
RA(1)	0.23	6790	14.76	2540
RA(2)	0.19	6930	11.18	2574
NLA(3)	0.48	5712	40.38	2316
NLA(5)	0.37	5627	27.43	2158
NLA(6)	0.34	5533	23.13	2137
LA(2, n)	0.50	6598	19.05	2378
LA(3, n)	0.95	6092	22.62	2238
LA(4, n)	0.61	6119	22.60	2359
LA(5, n)	0.60	6188	27.62	2473
LA(6, n)	1.07	6764	30.23	2693
LA(2, i)	0.35	7291	81.84	3248
LA(3, i)	0.38	6075	68.33	2261
LA(4, i)	0.39	5914	62.01	2215
LA(5, i)	0.49	5871	60.55	2213
LA(6, i)	0.56	5936	58.42	2269

In Table 2 we report the average number of nodes expanded in the last bucket (paths of length 508-512) over all maps as well as the total suboptimality over all paths and all maps. Suboptimality is measured as $100 \times (\frac{\text{actual path length}}{\text{optimal path length}} - 1)$, *i.e.*, the percentage difference in length. While there is some correlation between the nodes expanded by simple refinement and by PRA*, there are some differences as well. For instance, the sector abstraction expands fewer nodes (relative to the other abstractions) with simple refinement than with PRA*. This is due to the fact that simple refinement tends to produce paths with less suboptimality in the sector abstraction, and shorter paths are cheaper to refine. This is compared to the non-uniform line abstraction, which produces paths which are up to 80% longer than optimal.

Due to space concerns, we cannot reproduce the full graphs of the nodes expanded for all abstraction mechanisms here. However, we show the 5, 50 and 95th percentile curves for nodes expanded using PRA* using six of the different abstraction mechanisms in Figure 11. Some abstractions, like SA(3), produce a significantly wider spread of best-case and worst-case paths than the clique abstraction. These are percentile curves, so they are different than the average value which we report above. We also show optimality for several abstraction, comparing PRA* and simple refinement in Figure 12. These graphs show the worst quality path found, as well as the quality of the 99.5, 98, 95 and 50th percentile. The 50th percentile line is not visible for PRA* and falls just above the x-axis for simple refinement.

4.3 Predicting Total Work

We can now address the main point of our experimental results, which is to test whether the average number of nodes expanded, n , and the diameter of an abstraction, d or d_E , can be used to predict the total work needed to compute a path using an abstraction. For the moment, we just consider d_E .

First, recall that we are trying to minimize d_E and maximize n . In order to simplify our analysis, we look instead at the problem of minimizing both d_E and $\frac{1}{n}$. We normalize d and $\frac{1}{n}$ over all abstractions to the range $0 \dots 1$ and plot them in Figure 8. The size of the point for each algorithm scaled by the number of nodes expanded by PRA* on this abstraction. The figures are the same, the left one just includes labels of the data points. By looking at sequences of algorithms, one can see the effect of the abstraction parameters. The algorithms which abstract fewer nodes are found in the bottom right corner, while the algorithms which abstract more nodes are a time are in the top-left of the figure.

Because the total work is predicted to be correlated with both measures, we use the distance from each point in this chart to the origin of the graph (which attempts to minimize both measures equally) as a measure of the abstraction parameters. Since the parameters may not be equally weighted, we also considered the distance from the clique abstraction parameters as a secondary measure.

We plot these distances against the nodes expanded by simple refinement in Figure 9 and against the nodes expanded by PRA* in Figure 10. If these values are well-correlated, then the measures of n and d are accurate in predicting total work.

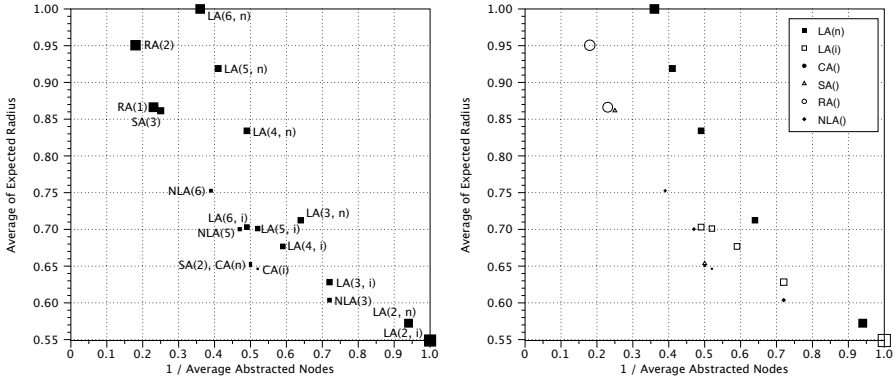


Fig. 8. The trade-off between the size of each abstraction and the diameter

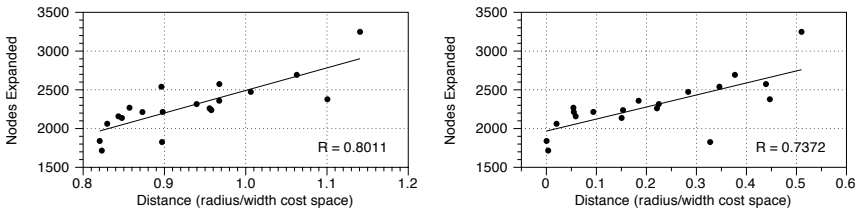


Fig. 9. The correlation between abstraction parameters and work done using simple refinement. The graph on the left measures the distance between the abstraction parameters relative to the origin, while the graph on the right measures the distance relative to the CA(n) parameters.

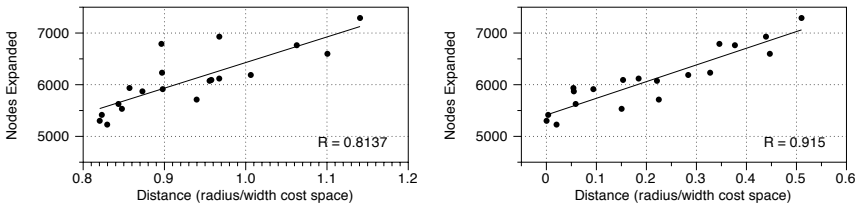


Fig. 10. The correlation between abstraction parameters and work done using PRA*. The graph on the left measures the distance between the abstraction parameters relative to the origin, while the graph on the right measures the distance relative to the clique abstraction parameters.

The best-fit line for search by the simple refinement algorithm has a correlation coefficient 0.80 using the distance from origin metric and 0.74 using the distance from the clique abstraction parameters. The best-fit line for work done by PRA* has a correlation of 0.81 when using the distance from the origin metric, and 0.92 when using the distance from the clique abstraction parameters.

Returning to the issue of whether it is better to use the maximum diameter, d , or the expected diameter, d_E , we ran the same predictions using both definitions of the diameter and distance. We found that the correlation between the best-fit line and the data was more accurate when using the expected diameter instead of the maximum diameter for both algorithms and distance measures.

5 Conclusions

We have shown that the total work done when using a graph abstraction for search and refinement can be predicted by two parameters, the diameter of abstract nodes, d , and the total number of nodes abstracted into each abstract node, n . Thus, the theoretical predictions of Holte *et al.* [2] are useful in practice as well. We have also shown that the clique abstraction's parameters are well suited for minimizing computation, particularly for PRA*.

Additionally, we have introduced a new method for computing the expected diameter of an abstract node, d_E , and have shown that this is more accurate than using the maximum diameter, d . Our analysis has also highlighted how small changes in how an abstraction is built can influence the abstraction measures, n and d . This was shown in the differences between uniform and non-uniform clique and line abstractions.

There are two specific areas for future research. First, we would like to expand these results beyond the pathfinding domain. Secondly, we would like to better understand the manner in which suboptimality is affected by the choice of abstraction and how it influences the predictions of how much work must be done. While we have seen that there is some influence, we have yet to describe this influence in detail.

References

1. Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2), 100–107 (1968)
2. Holte, R.C., Mkdmi, T., Zimmer, R.M., MacDonald, A.J.: Speeding up problem solving by abstraction: A graph oriented approach. *Artificial Intelligence* 85, 321–361 (1996)
3. Botea, A., Müller, M., Schaeffer, J.: Near optimal hierarchical path-finding. *Journal of Game Development* 1(1), 7–28 (2004)
4. Sturtevant, N.R., Buro, M.: Partial pathfinding using map abstraction and refinement. In: *AAAI*, pp. 1392–1397 (2005)
5. Fernandez, A., Gonzalez, J.: *Multi-Hierarchical Representation of Large-Scale Space*. Kluwer Academic Publishers, Dordrecht (2001)
6. Yang, Q., Tenenber, J., Woods, S.: On the implementation and evaluation of ABT weak. *Computational Intelligence Journal* 12, 295–318 (1996)

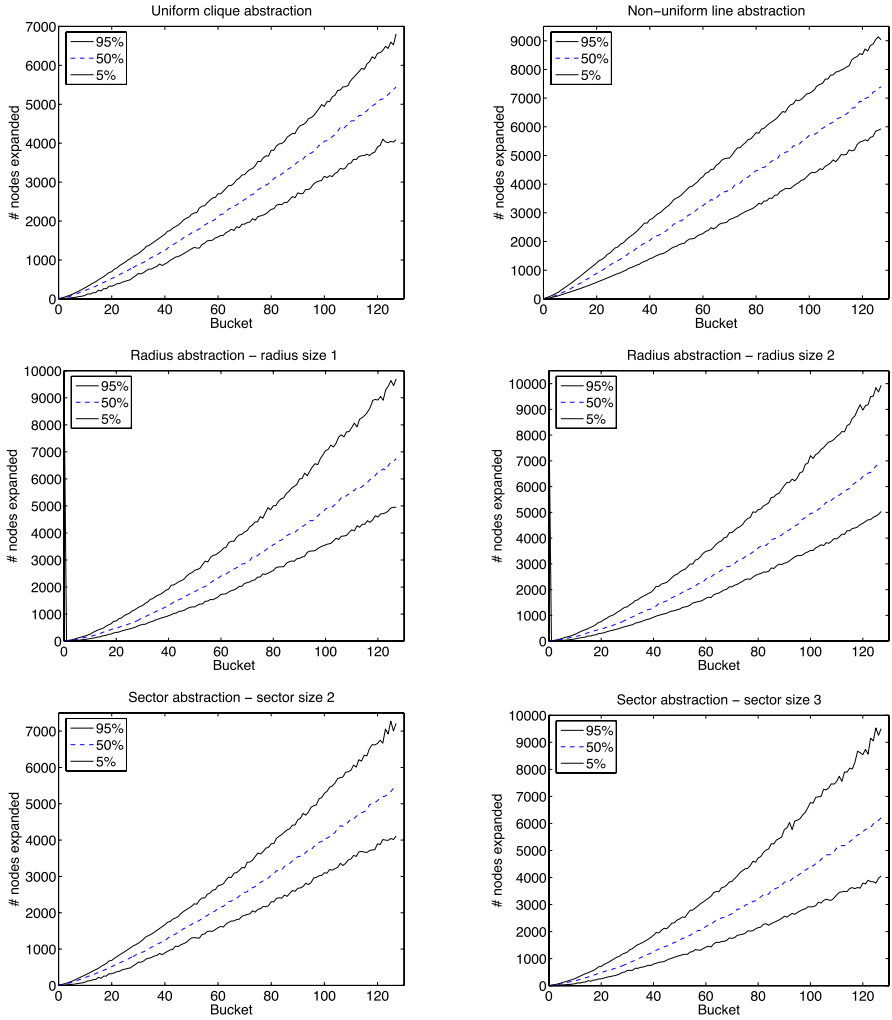


Fig. 11. Nodes expanded by 6 of the different abstraction mechanisms

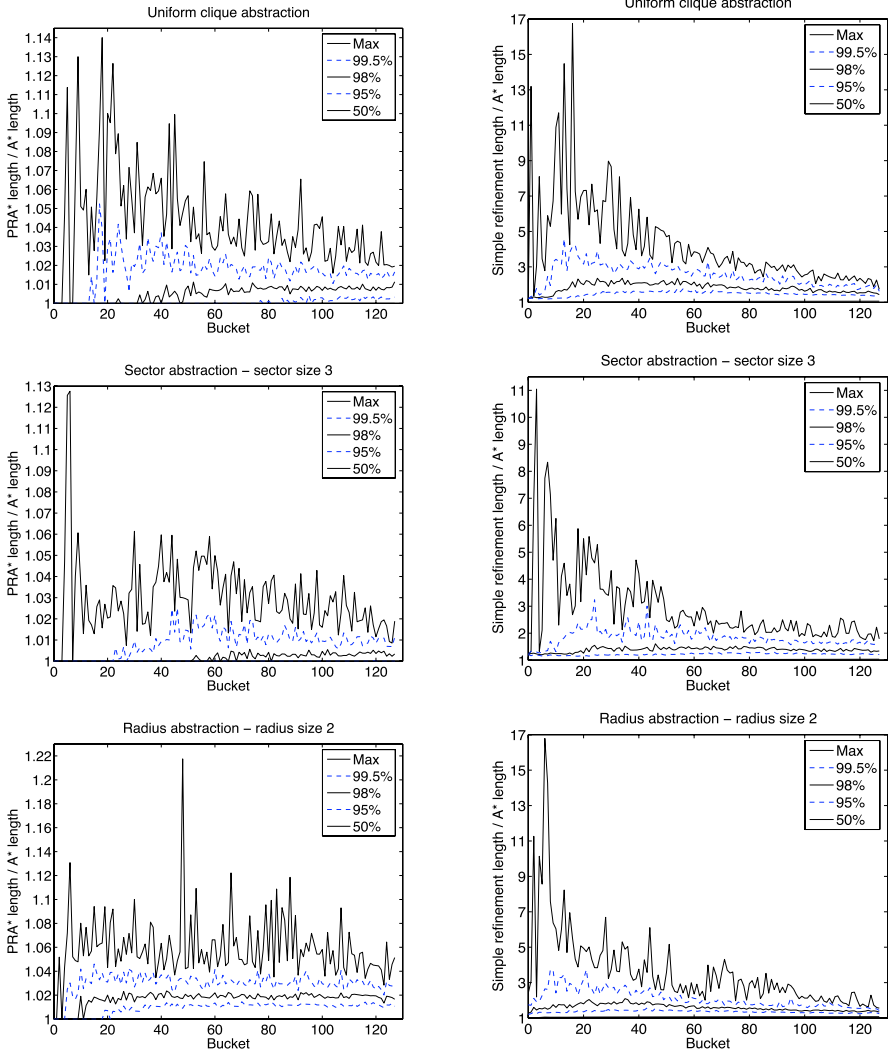


Fig. 12. Path optimality using 3 of the different abstraction mechanisms and $PRA^*(\infty)$ [left] versus simple refinement [right]

Solving Difficult SAT Instances Using Greedy Clique Decomposition*

Pavel Surynek

Charles University
Faculty of Mathematics and Physics
Malostranské náměstí 2/25, 118 00 Praha 1, Czech Republic
surynek@ktiml.mff.cuni.cz

Abstract. We are dealing with solving difficult SAT instances in this paper. We propose a method for preprocessing SAT instances (CNF formulas) by using consistency techniques known from constraint programming methodology and by using our own consistency technique based on clique decomposition of a graph representing conflicts in the input formula. If the clique decomposition is of a good quality (cliques are appropriately large) it then allows us to make a strong reasoning over the SAT instance, which can in some cases even decide the satisfiability of the SAT instance without search. We implemented our preprocessing method in C++ and compared it with several state-of-the-art SAT solvers on selected difficult SAT instances. The result was a speedup in the order of magnitude compared to the tested SAT solvers.

Keywords: SAT, search, consistency, clique, difficult instances.

1 Introduction

The source of inspiration for this paper was a recent work [28] on artificial intelligence planning problems [3]. We exploit the newly developed techniques proposed in [28] for solving *Boolean satisfiability problems (SAT)*. In [28] the problem of finding supporting actions for a goal in the AI planning context is studied. The problem is called a *supports problem* in short. This is some kind of an important sub-problem which must be solved many times when solving AI planning problems using the *planning graphs* [6]. It was shown that the supports problem is NP-complete. In doing so a conversion of an instance of the SAT problem to the instance of the supports problem was used [28]. This proof uncovered some interesting similarities between the SAT problem and the supports problem. Strictly speaking the similarities itself are neither interesting nor useful. They become more interesting after connecting them with the new method for solving supports problems based on a greedy clique decomposition which was also proposed in the mentioned work. The positive experience made with

* This work is supported by the Czech Science Foundation under the contracts 201/07/0205 and 201/05/H014. I would like to thank anonymous reviewers for many useful comments and corrections. I also would like to thank my advisor Roman Barták for valuable discussions.

the method on planning problems and the observed similarity lead us to the idea of adapting the technique of the greedy clique decomposition to solve SAT problems.

Boolean formula satisfaction problems and SAT solving techniques play an extremely important role in theoretical computer science as well as in practice. The question of whether there exist a complete polynomial time SAT solver is a key question for theoretical computer science and is open for many years (the *P vs. NP problem*) [7]. On the other hand the practical use of SAT problems and SAT solvers in real life applications is also very intensive. Applications of SAT solving techniques range from microprocessor verification [30] and field-programmable gate array design [23] to solving AI planning problems by translating them into Boolean formulas [17].

An excellent performance breakthrough was done in solving SAT problems over the past years. Thanks to new algorithms and implementation techniques focused on real life SAT problems many of the today's benchmark problems [18, 25] are solved by state-of-the-art solvers [11, 12, 14, 15, 21, 27] in time proportional to the size of the input. It seems that the difficulty of many SAT benchmark problems consists in their size only. A lot of smaller benchmark problems are solved in real-time by today's state-of-the-art SAT solvers. The observation that can be deduced upon these facts is that there is almost no chance to compete with the best SAT solvers by a newly written SAT solver on these problems. That is why we are concentrating on difficult instances of SAT problems only, where the word difficult means difficult for today's state-of-the-art SAT solvers.

A very valuable set of difficult (in the mentioned sense) problems was collected by Aloul [1]. Although these problems are small in the length of the input formula they are difficult to be answered. The detailed discussion about hardness of these problems is given in [2]. One of the aspects of problem difficulty is that these problems are mostly unsatisfiable (and this fact is well hidden in the problem). The solver cannot guess a solution using its advanced techniques and heuristics in such a case and it must really perform some search in order to prove that there is no solution. In the case of a positive answer the satisfying valuation of variables serves a witness (of small size) certifying existence of at least one solution. If the solver obtains (possibly by guessing) a witness its task is finished. In contrast to this, there is no such small witness in the case of a negative answer so the search must be performed.

Our contribution to solving SAT problems consists of preprocessing and reformulating of the input Boolean formula in the *CNF* (*conjunctive normal form* - conjunction of disjunctions). The result of this processing is the answer whether the input formula is unsatisfiable or a new formula (hopefully simpler) with the same set of satisfying valuations as that of the input one. If the input formula is not decided by the preprocessing phase then the preprocessed formula is sent to the SAT solver of the user's choice. The idea behind this process is to make the task for the SAT solver easier by deciding the input formula within the fast preprocessing phase or by providing an equivalent but simpler formula to the SAT solver. Experiments showed that the solving process over the above mentioned difficult SAT benchmarks speeds up by the order of magnitude after using our approach.

The reformulation within the preprocessing phase itself is simple. We are viewing the input Boolean formula in CNF as a graph (with vertices and edges). For each *literal* (variable or its negation) of the input formula we consider a vertex and for each conflict between literals we consider an edge. Conflicting literals are those that cannot

be both satisfied in a single valuation of variables, for example positive and negative literals of the same variable are conflicting. Generally, a set of literals of a formula is conflicting if the formula entails that at most one of the literals can be *true*. To be able to use our reasoning based on the clique decomposition we need a graph with appropriately large *complete sub-graphs* (cliques). That is, we need some kind of a good approximation of the sets of conflicting literals. Unfortunately the graph arising from the above interpretation of the Boolean formula in *CNF* is rather sparse (the largest clique is of size 2). That is why we apply further inference by which we deduce more conflicts between the literals and which allow us to introduce more edges into the graph. We are using singleton arc-consistency [5] as the inference technique for deducing new edges.

Having the graph constructed from the input *CNF* formula, a clique decomposition of this graph is found by a greedy algorithm (we do not need an optimal clique decomposition; we need just some of the reasonable quality). The important property of the constructed clique decomposition is that at most one literal from each clique can be assigned the value *true*. In this situation we perform some kind of literal contribution counting to rule out the literals that can never be *true*. To do this, the maximum number of satisfied clauses by literals of each clique is calculated. Then a literal of a certain clique can be ruled out if the literals from the other cliques together with the selected literal do not satisfy enough clauses to satisfy the input formula.

Although this problem reformulation seems weak it provides a strong reasoning about the dependencies among clauses of the *CNF* Boolean formula and about the effect of the selection of a value for a variable on the overall satisfiability of the formula. Moreover if all the literals are ruled out during the preprocessing phase the input formula is obviously unsatisfiable. Experimental evaluation showed that this happen frequently on difficult SAT problems. For other cases a new formula in the *CNF* equivalent to the input formula is produced. The new formula is constructed from the original one by adding clauses that capture all the dependencies inferred by the initial singleton arc-consistency stage and by the literal contribution counting based on the clique decomposition.

The paper is organized as follows. A detailed formal description of the reformulation of a SAT instance using the greedy clique decomposition is given in section 2. The subsequent section 3 is devoted to some experimental comparison of our approach with the existing state-of-the-art SAT solvers. We are discussing the contribution of our method within this section too. Finally we put our work in relation to similar works in the field of Boolean satisfiability and we propose some future research directions of the studied topic.

2 SAT Reformulation Using Greedy Clique Decomposition

We will formally describe the details of the process of SAT problem reformulation in this section. Let $B = \bigwedge_{i=1}^n \bigvee_{j=1}^{m_i} x_j^i$ be the input Boolean formula in *CNF* where x_j^i is a literal (variable or its negation) for all possible i and j . A sub-formula $\bigvee_{j=1}^{m_i} x_j^i$ of the input formula B for every possible i is called a clause. The i th clause of the formula B will be denoted as C_i in the following paragraphs. As it was mentioned in the introduction, the basic idea of the SAT problem reformulation consists in viewing the

input formula as an undirected graph in which the internal structure of the formula is captured in some way. To be more particular the graph will capture the pairs of conflicting literals and it will be constructed in several stages.

2.1 Inference of Conflicting Literals

We start by the construction of an undirected graph $G_B^1 = (V_B^1, E_B^1)$ which will represent trivially conflicting literals. The graph will be called a *graph of trivial conflicts*. The graph G_B^1 will then undergo some further inference process by which the additional conflicts will be inferred. We will denote the resulting undirected graph as $G_B^2 = (V_B^2, E_B^2)$ and call it an *intermediate graph of conflicts*.

The construction of the undirected graph G_B^1 is simple. A vertex is introduced into the graph G_B^1 for each literal occurring in the formula B , that is $V_B^1 = \bigcup_{i=1}^n \bigcup_{j=1}^{m_i} x_j^i$ (notice that $|V_B^1|$ is typically smaller than the length of the formula, since literals may occur many times in the formula while only once in the graph). The construction of the set of edges E_B^1 is also straightforward. An edge $\{x_j^i, x_l^k\}$ is introduced into the graph G_B^1 if the literals x_j^i and x_l^k are trivially conflicting, that is if one is a variable v and the other is $\neg v$ for some Boolean variable v . The graph G_B^1 is finished by performing the above step for all possible pairs of conflicting literals. The interpretation of the graph of conflicts is that if a literal corresponding to a vertex is selected to be assigned the value *true* all literals corresponding to the neighboring vertices must be assigned the value *false*.

An example graph resulting from the described process over a selected benchmark problem is shown in the left part of figure 1. The resulting graph is visibly sparse, since there are edges only between the literals of the same variable. Hence it is not a good starting point for our method and a further inference mechanism for discovering more conflicting pairs of literals (more edges for the graph) must be applied. This further inference mechanism takes the already constructed graph G_B^1 and augments it by adding new edges. The result of this stage is an intermediate graph of conflicts G_B^2 .

The process of construction of graph G_B^2 exploits techniques known from standard SAT resolution approaches and from *constraint programming* [9] - *unit propagation* [10, 31], *arc-consistency (AC)* [20] and *singleton arc-consistency (SAC)* [5]. Before describing the construction of the graph G_B^2 let us recall a modification of notions. We modify the above concepts slightly for the SAT domain to prepare them for our purposes. The following definitions assume the input formula B in CNF and a corresponding graph of conflicts G_B (for example the graph G_B^1 expressing the trivial conflicts).

Definition 1 (Arc-consistency in SAT instance w.r.t. the graph of conflicts). Consider two clauses C_i and C_k for $i, k \in \{1, 2, \dots, n\}$, $i \neq k$ of the formula B . A literal x_j^i ($j \in \{1, 2, \dots, m_i\}$) from the clause C_i is *supported* by the clause C_k with respect to the given graph of conflicts G_B if there exists a literal x_l^k ($l \in \{1, 2, \dots, m_k\}$) from the clause C_k , such that the literals x_j^i and x_l^k are not in a conflict with respect to the graph G_B (not connected by an edge). An ordered pair of clauses (C_i, C_k) of the formula B is called an *arc* in this context. An arc (C_i, C_k) for some $i, k \in \{1, 2, \dots, n\}$

is *consistent* (or *arc-consistent*) with respect to the graph of conflicts G_B if all the literals of the clause C_i are supported by the clause C_k with respect to the graph of conflicts G_B . The formula B is called *arc-consistent* with respect to the graph of conflicts G_B if all the arcs (C_i, C_k) for all $i, k = 1, 2, \dots, n$ are arc-consistent with respect to the graph of conflicts G_B . \square

Let us note that our definition is based on a dual view of the satisfaction problem. That is, we use the clauses of the formula as the CSP variables [9] instead of the original Boolean variables. Having these CSP variables, (CSP) constraints necessary for the definition of arc-consistency arise naturally.

The reason for the definition of arc-consistency is that the literals which are not supported according to the definition cannot be assigned the value *true* (this means that the corresponding variable cannot be assigned the value *false* in the case of a negative literal). So the solver can rule out such literals from further attempts to assign them the value *true*, which may reduce the size of the search space. Notice that the definition has the graph of conflicts G_B as a parameter. It is possible to put any correct graph of conflicts as a parameter of this definition, whereas correct means, that if $\{y, z\}$ is the edge of the graph then $B \Rightarrow y \neq z$ must be a tautology. This is obviously true for the graph of trivial conflicts G_B^1 . Notice also that if we use the graph of trivial conflicts G_B^1 the definition becomes identical to unit propagation [10, 31].

Having the Boolean formula B the question is how to make it arc-consistent with respect to the given graph of conflicts. For this purpose we adopt techniques developed in constraint programming and by SAT community, namely the arc-consistency enforcing algorithms [9, 20] and unit propagation [10, 31]. There is a great variety of such algorithms, however their common feature is the search for supports for every value (literal) which is suspected of not being supported. The main difference among these algorithms is the efficiency of the search for supports. If an unsupported literal is detected it is ruled out. Ruling out an unsupported literal may cause that some other literal loses its only support. This chain-like propagation of changes continues until a stable state is reached. For purposes of the SAT domain this propagation process is usually augmented by an additional simplification rule. If the consistency enforcing algorithm detects that within some clause there is only one literal that can be selected to be *true*, it is fixed to value *true* and the corresponding clause is cut out from further reasoning (this is exactly the simplification rule from unit propagation).

Unfortunately the defined arc-consistency over Boolean formulas in the CNF form is too weak to infer significantly more conflicts than that are already present in the graph of trivial conflicts. Therefore we need to make the consistency stronger. Perhaps the simplest way to do this is to make the selected consistency technique *singleton* [5]. The following definition again assumes the Boolean formula B and the corresponding graph of conflicts G_B (again the graph of trivial conflicts G_B^1 can be used).

Definition 2 (Singleton arc-consistency in a SAT instance w.r.t. the graph of conflicts). A literal x_l^k ($l \in \{1, 2, \dots, m_k\}$) from a clause C_k for $k \in \{1, 2, \dots, n\}$ of the formula B is *singleton arc-consistent* with respect to the given graph of conflicts G_B

if the formula obtained from B by replacing the clause C_k by the literal x_l^k (the resulting formula is $(\bigwedge_{i=1}^{k-1} \bigvee_{j=1}^{m_i} x_j^i) \wedge x_l^k \wedge (\bigwedge_{i=k+1}^n \bigvee_{j=1}^{m_i} x_j^i)$) is arc-consistent with respect to the graph of conflicts G_B . \square

Unsupported literals in the formula modified by replacing the clause C_k by the literal x_l^k are in conflict with the literal x_l^k . This is quite intuitive, the selection of the literal x_l^k to be assigned the value *true* rules out some other literals. Hence these literals are in conflict with the selected literal x_l^k . Having singleton arc-consistency we are ready to infer new edges for the graph of conflicts.

The intermediate graph of conflicts G_B^2 is constructed from the graph of trivial conflicts G_B^1 in the following way. Initially the graph G_B^2 is identical to the graph G_B^1 , that is we start with the initialization $V_B^2 \leftarrow V_B^1$ and $E_B^2 \leftarrow E_B^1$. Then for every literal $y \in V_B^2$ singleton arc-consistency with respect to the graph of conflicts G_B^1 is enforced. If the consistency discovers some unsupported literals, say literals z_1, z_2, \dots, z_m , edges $\{y, z_i\}$ for all $i=1, 2, \dots, m$ are added into the set of edges E_B^2 .

An example of the resulting graph of conflicts is shown in the right part of the figure 1. It is constructed from the original graph of trivial conflicts from the left part of the figure 1. The required complete sub-graphs of the graph are clearly visible.

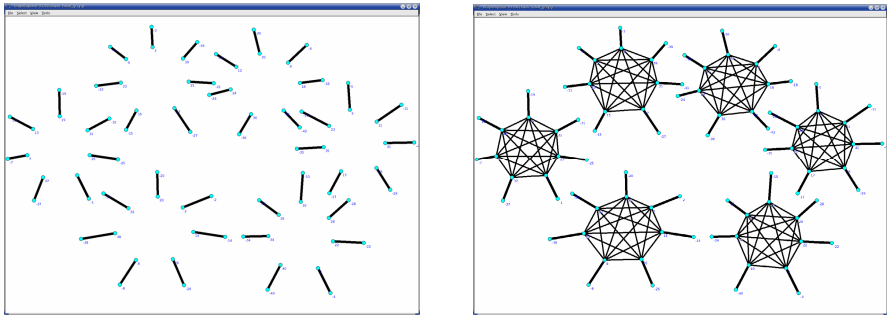


Fig. 1. The left part of the figure shows a graph of trivial conflicts for the SAT benchmark problem pigeon-hole principle number 6 (hole06.cnf). Vertices represents literals, edges are between pairs of positive and negative literals of the same variable. The right part of the figure shows an intermediate graph of conflicts inferred from the original graph of the left by singleton arc-consistency. The graph contains edges from the original graph plus the inferred edges. Six complete sub-graphs each containing seven vertices are clearly visible and can be found by a simple greedy algorithm.

The described process of inference of conflicting literals is relatively generic. Both different initial graphs of trivial conflicts as well as different consistency techniques than arc-consistency and singleton arc-consistency for inference of new edges can be used. Both entities, graphs and consistency techniques, may be considered as parameters of the method.

2.2 Greedy Clique Decomposition and Literal Contribution Counting

To deduce yet more information from the graph of conflicts $G_B^2 = (V_B^2, E_B^2)$ a clique decomposition of the graph is constructed. Formally, a partition of vertices $V_B^2 = K_1 \cup K_2 \cup \dots \cup K_s$ such that each set of vertices K_i for $i=1,2,\dots,s$ induces a clique over the set of edges E_B^2 and $K_i \cap K_j = \emptyset$ for all $i, j=1,2,\dots,s$ & $i \neq j$. Let E_{K_i} denotes the set of edges induced by the clique K_i , let E_R denotes the set of edges outside the clique decomposition, that is $E_R = E_B^2 - \bigcup_{i=1}^s E_{K_i}$. Our inference method based on literal contribution counting performs best if cliques of the decomposition are as large as possible (that is s must be as small as possible) and the size of E_R is as small as possible. The better the quality of the decomposition is the stronger results are produced by our inference method. Since the problem of finding the optimal clique decomposition with respect to the above criterion is obviously *NP*-complete on a general graph [16], we cannot afford to construct the optimal decomposition and we must abandon this requirement. Nevertheless experiments showed that the simple greedy algorithm can find a clique decomposition of acceptable quality (with respect to clique sizes and the number of edges outside the decomposition).

Our greedy algorithm for finding a clique decomposition is based on the standard greedy algorithm for finding the largest clique. The main loop of the greedy algorithm repeatedly finds a largest clique. The largest clique is found in the following way. A vertex of the highest degree is found in the graph and it is added to the constructed clique which is empty at the beginning. Then the graph is restricted on the neighborhood of the selected vertex and a vertex of the highest degree in this neighborhood is selected as second. Then the graph is again restricted on the neighborhood of these two vertices (that is the considered vertices are neighbors of both the first and the second selected vertex) and the algorithm continues until the neighborhood of selected vertices is empty. The constructed clique and its neighborhood are removed from the graph and the next clique is constructed. This main loop continues until the graph is empty.

The above described greedy algorithm performed over the graph from the right part of the figure 1 finds the clique decomposition consisting of six cliques of size seven. The fact that at most one literal from a clique can be selected to be assigned the value *true* is used in our inference method.

For the following definitions we assume a Boolean formula $B = \bigwedge_{i=1}^n \bigvee_{j=1}^{m_i} x_j^i$ and the corresponding clique decomposition $V_B^2 = K_1 \cup K_2 \cup \dots \cup K_s$ of the intermediate graph of conflicts $G_B^2 = (V_B^2, E_B^2)$. Next let $I \subseteq \{1,2,\dots,n\}$ be a set of indexes of some clauses of the formula B . The set I defines a sub-formula B_I of the formula B , where $B_I = \bigwedge_{i \in I} C_i$.

Definition 3 (Literal contribution). A *contribution of a literal y* to the sub-formula B_I is defined as the number of clauses of B_I in which the literal y occurs and it is denoted as $c(y, I)$. \square

Definition 4 (Clique contribution). A *contribution of a clique $K \in \{K_1, K_2, \dots, K_s\}$* to the sub-formula B_I is defined as $\max_{y \in K} (c(y, I))$ and it is denoted as $c(K, I)$. \square

The concept of clique contribution is helpful when we are trying to decide whether it is possible to satisfy the sub-formula B_i using the literals from the clique decomposition. If for instance $\sum_{i \in I} c(K_i, I) < |I|$ holds then the sub-formula B_i cannot be satisfied and hence also B cannot be satisfied. Moreover we can handle a more general case as it is described in the following definitions.

Definition 5 (Clique-consistent literal). A literal $y \in K_i$ for $i \in \{1, 2, \dots, n\}$ is said to be *clique-consistent with respect to the sub-formula B_i* if $\sum_{j \in I \ \& \ j \neq i} c(K_j, I) \geq |I| - c(y, I)$. □

Definition 6 (Clique-consistent formula). A formula B is *clique-consistent with respect to the sub-formula B_i* if all the literals of the formula B are clique-consistent with respect to B_i . □

It is easy to see that a clique-inconsistent literal with respect to some sub-formula of B cannot be selected to be assigned the value *true*. Thus such literals can be ruled out from further reasoning. The proof of this claim is provided in the technical report [28]. In addition, this type of consistency is strictly stronger than the discussed unit propagation, arc-consistency and singleton arc-consistency. The proof of this claim is again given in [28].

The remaining question is how to select the described sub-formulas B_i of B which are used for computation of the clique-inconsistent literals. This selection is crucial for the strength of the proposed clique-consistency. It is clear that we need to rule out as many as possible inconsistent literals. As it is impossible to compute the defined consistency with respect to all such sub-formulas of B , because there are too many sub-formulas, we need to select a subset of them carefully. The experiments carried out in [28] showed that a good strength of the clique-consistency can be obtained by selecting clauses into the sub-formula B_i which have the same number of literals. More precisely, we use sub-formulas $B_{I_r} = \bigwedge_{i \in I_r} C_i$ of B , where $I_r = \{i \in \{1, 2, \dots, n\} \mid m_i = r\}$ for all possible $r \in \mathbb{N}$ for which B_{I_r} is not empty (we suppose that a clause of B does not contain an individual literal more than once). Let us note that we do not know whether this selection is the best possible.

Theorem 1 (Complexity of clique-consistency enforcing algorithm). *There exists a polynomial time algorithm for enforcing clique-consistency with respect to a sub-formula of a given input formula.* ■

The proof of this theorem can be found in [28]. Having such an algorithm it is possible to extend it for multiple sub-formulas B_{I_r} simply by running the algorithm for each $r \in \mathbb{N}$ for which B_{I_r} is non-empty. Since r is proportional to the size of the input the, resulting algorithm is also polynomial.

2.3 Output of the Reformulation Process

At this point everything is ready to introduce the final step of our reformulation method. We will be constructing a modified formula β which is initially set to be identical to B . We will further preprocess B by the singleton version of the defined clique-consistency. Conflicts inferred by this further preprocessing will be stored in a

new graph of conflicts $G_B^3 = (V_B^3, E_B^3)$ which is initially set to be the same as the graph G_B^2 . The graph G_B^3 will be called a *final graph of conflicts* in this context.

Singleton clique-consistency is computed in the following way. For each literal y of the input formula B we enforce clique-consistency for the formula obtained from B by selecting a literal y to be assigned the value *true*. More precisely, clauses containing y are removed and the negation of the literal y is removed from remaining clauses of B (removal of a literal x_k^i from the clause $C_i = \bigvee_{j=1}^{m_i} x_j^i$ of the formula B is defined as replacement of the clause C_i by the clause $(\bigvee_{j=1}^{k-1} x_j^i) \vee (\bigvee_{j=k+1}^{m_i} x_j^i)$). The clique-consistency is then enforced for the resulting formula. Some literals may be found inconsistent during consistency enforcing. These literals are in conflict with the literal y . If for some clause all its literals are found inconsistent with y then the literal y cannot be selected to be *true* and a new clause $\neg y$ is added to β ($\beta \leftarrow \beta \wedge \neg y$). Otherwise the conflicting literals are stored in the graph of conflicts G_B^3 as new edges (that is, if the literal y is in conflict with the literal z , the edge $\{y, z\}$ is added to G_B^3).

If for some clause it is discovered by the clique-consistency that none of its literals can be assigned the value *true* the process terminates with the answer that the formula B cannot be satisfied. This outcome is ensured by the correctness of the method. Our experiments showed that this situation is the most successful case, because an answer to the satisfiability is obtained in polynomial time without further expensive search for a solution.

If the process does not terminate with the negative answer then all the edges of the graph of conflicts G_B^3 are translated into new clauses of the formula β . That is, for every edge $\{y, z\} \in E_B^3$ we add a clause $y \vee \neg z$ into the formula β ($\beta \leftarrow \beta \wedge (y \vee \neg z)$). The resulting formula β is equivalent with the original input formula B . Notice that the conflicts inferred by the preceding reformulation stages are also reflected in the formula β , since the graph G_B^3 subsumes the preceding graphs of conflicts G_B^1 and G_B^2 . The formula β is finally sent to the SAT solver of the user's choice. Justification of this step is provided by the following corollary of the correctness of the clique-consistency.

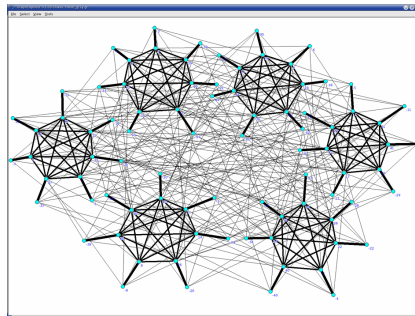


Fig. 2. A final graph of conflicts for the SAT benchmark problem pigeon-hole principle number 6 (hole06.cnf). The graph contains edges from the intermediate graph of conflicts from figure 1 plus the edges inferred by singleton clique-consistency.

Corollary 1 (Correctness of reformulation). *The formula β resulting from the described preprocessing has the same set of satisfying valuations as the original formula B .* ■

The graph of conflicts G_B^3 resulting from processing the intermediate graph of conflicts G_B^2 for the SAT benchmark problem from figure 1 is shown in figure 2.

3 Experimental Results

We chose three state-of-the-art SAT solvers for comparison with our reformulation method. The SAT solvers of our choice were zChaff [14, 21], HaifaSAT [15, 27] and MiniSAT (a version with SATElite preprocessing integrated) [11, 12] (we used the latest available versions to the time of writing this paper). Our choice was guided by the results of several last SAT competitions [18, 25] in which these solvers belonged to the winners. The secondary guidance was that complete source code (in C/C++) for all these solvers is available on web pages of their authors. As we implemented our method in C++ too, this fact allowed us to compile all source codes by the same compiler with the same optimization options which guarantees more equitable conditions for the comparison (a complete source code implementing our method in C++ available at the web page: <http://ktiml.mff.cuni.cz/~surynek/software/ssat/ssat.html>). All the tests were run on the machine with two AMD Opteron 242 processors (1600 MHz) with 1GB of memory under Mandriva Linux 10.2. Our method as well as the listed SAT solvers were compiled by the gcc compiler version 3.4.3 with options provided maximum optimization for the target testing machine (-O3 -mtune=opteron). Although the testing machine has two processors no parallel processing was used.

3.1 Difficult SAT Instances Selected for Experiments

The testing set consisted of several difficult unsatisfiable SAT instances. This set of benchmark problems was collected by Aloul [1] and it is provided at his research web page. The details about hardness and construction of these instances are discussed in [2], but let us briefly introduce the problems.

Pigeon Hole Instances. [*hole*] This is a standard SAT benchmark encoding the pigeon hole principle problem. The problem asks whether it is possible to place $n + 1$ pigeons in n holes without two pigeons being in the same hole. The problem is obviously unsatisfiable. We used six instances of this problem ranging from 6 to 12 holes.

Randomized Urquhart Instances. [*urq*] This set of benchmark problems contains several artificially constructed hard unsatisfiable instances. More details about these problems are provided in [29]. In addition, the problems were randomized for our testing purposes. We used four instances of the problems of this type.

Table 1. Experimental comparison of three SAT solvers over the selected difficult benchmark SAT instances. We used the timeout of 10.0 minutes (600.00 seconds) for all the tests.

Instance	Satisfiable	Number of variables / number of clauses	MiniSAT (seconds)	zChaff (seconds)	HaifaSAT (seconds)
chnl10_11	unsat	220/1122	34.30	7.54	> 600.00
chnl10_12	unsat	240/1344	101.81	9.11	> 600.00
chnl10_13	unsat	260/1586	200.30	11.47	> 600.00
chnl11_12	unsat	264/1476	> 600.00	33.49	> 600.00
chnl11_13	unsat	286/1472	> 600.00	187.08	> 600.00
chnl11_20	unsat	440/4220	> 600.00	329.57	> 600.00
urq3_5	unsat	46/470	95.04	> 600.00	> 600.00
urq4_5	unsat	74/694	> 600.00	> 600.00	> 600.00
urq5_5	unsat	121/1210	> 600.00	> 600.00	> 600.00
urq6_5	unsat	180/1756	> 600.00	> 600.00	> 600.00
hole6	unsat	42/133	0.01	0.01	0.01
hole7	unsat	56/204	0.09	0.04	0.02
hole8	unsat	72/297	0.49	0.23	0.94
hole9	unsat	90/415	3.64	1.46	478.16
hole10	unsat	110/561	39.24	7.53	> 600.00
hole11	unsat	132/738	> 600.00	32.36	> 600.00
hole12	unsat	156/949	> 600.00	372.18	> 600.00
fpga10_11	unsat	220/1122	44.77	12.58	> 600.00
fpga10_12	unsat	240/1344	119.26	33.82	> 600.00
fpga10_13	unsat	260/1586	362.24	76.15	> 600.00
fpga10_15	unsat	300/2130	> 600.00	274.84	> 600.00
fpga10_20	unsat	400/3840	> 600.00	546.00	> 600.00
fpga11_12	unsat	264/1476	> 600.00	55.70	> 600.00
fpga11_13	unsat	286/1742	> 600.00	237.54	> 600.00
fpga11_14	unsat	308/2030	> 600.00	> 600.00	> 600.00
fpga11_15	unsat	330/2340	> 600.00	> 600.00	> 600.00
fpga11_20	unsat	440/4220	> 600.00	> 600.00	> 600.00

Table 2. Experimental comparison of three SAT solvers with the method using clique- consistency over the selected difficult benchmark SAT instances. Again timeout of 10.0 minutes (600.00 seconds) for all the tests was used.

Instance	Decided by preprocessing	Cliques (count x size)	Decision (seconds)	Speedup ratio w.r.t. MiniSAT	Speedup ratio w.r.t. zChaff	Speedup ratio w.r.t. HaifaSAT
chnl10_11	yes	20 x 11	0.43	79.76	17.53	> 1395.34
chnl10_12	yes	20 x 12	0.60	169.68	8.51	> 1000.00
chnl10_13	yes	20 x 13	0.78	256.79	14.70	> 769.23
chnl11_12	yes	22 x 12	0.70	> 857.14	47.84	> 857.14
chnl11_13	yes	22 x 13	0.92	> 652.17	203.34	> 652.17
chnl11_20	yes	22 x 20	5.74	> 104.42	57.41	> 104.42
urq3_5	no	47 x 2	130.15	0.73	N/A	N/A
urq4_5	no	73 x 2	> 600.00	N/A	N/A	N/A
urq5_5	no	120 x 2	> 600.00	N/A	N/A	N/A
urq6_5	no	179 x 2	> 600.00	N/A	N/A	N/A
hole6	yes	6 x 7	0.01	1.0	1.0	1.0
hole7	yes	7 x 8	0.02	4.5	2.0	1.0
hole8	yes	8 x 9	0.04	12.25	5.75	23.5
hole9	yes	9 x 10	0.08	45.5	18.25	5977.00
hole10	yes	10 x 11	0.13	301.84	57.92	> 4615.38
hole11	yes	11 x 12	0.20	> 3000.00	161.8	> 3000.00
hole12	yes	12 x 13	0.30	> 2000.00	1240.6	> 2000.00
fpga10_11	yes	20 x 11	0.46	97.32	27.34	> 1304.34
fpga10_12	yes	20 x 12	0.64	186.34	52.84	> 937.50
fpga10_13	yes	20 x 13	0.84	431.23	90.65	> 714.28
fpga10_15	yes	20 x 15	1.39	> 431.65	197.72	> 431.65
fpga10_20	yes	20 x 20	4.72	> 127.11	115.67	> 127.11
fpga11_12	yes	22 x 12	0.76	> 789.47	73.28	> 789.47
fpga11_13	yes	22 x 13	1.01	> 594.05	235.18	> 594.05
fpga11_14	yes	22 x 14	1.30	> 461.53	> 461.53	> 461.53
fpga11_15	yes	22 x 15	1.67	> 359.28	> 359.28	> 359.28
fpga11_20	yes	22 x 20	5.96	> 100.67	> 100.67	> 100.67

Field Programmable Gate Array Routing Instances. [*fpga*, *chnl*] This benchmark problem resembles the pigeon hole problem. The question is whether it is possible to route n connections through m tracks provided by the field programmable gate array component. If $n > m$ the problem cannot be satisfied. We used sixteen unsatisfiable instances of this problem for various number of required routes and connections. Two different encodings of the problem are used - denoted *fpga* and *chnl*. More details about the encoding of this problem are provided in [23].

For each benchmark SAT instance we measured the overall time necessary to decide its satisfiability. The results are shown in table 1 and table 2. The speedup obtained by using our method compared to a selected SAT solver is also shown.

3.2 Effect of Problem Reformulation

As it is evident from our experiments the proposed method brings significant improvement in terms of time necessary for the decision of the selected difficult benchmark problems (Pigeon hole, FPGA routing instances). The improvements are in the order of magnitude in comparison to all tested state-of-the-art SAT solvers. It seems that the improvement on selected benchmarks is exponential with respect to the best tested SAT solver. The conclusion is that there is still a space to improve SAT solvers. However, the domain of the improvement is more likely in the difficult instances of SAT problems which are typically unsatisfiable. It is also evident that the clique-consistency is not an universal method for difficult SAT instances. There is no improvement on instances where no cliques of reasonable size are found (randomized Urquhart instances). The interesting feature of the tested SAT instances is that they contain cliques of the same size. This may be accounted to the symmetrical formulation of the problems.

In our further experiments we also performed the comparison with the RSAT solver [24]. The results were very similar in the sense that the solver does not cope well with these problems. Unfortunately the solver is provided without the source code so we do not consider this test as a relevant one. Another SAT solver which worth consideration for our tests (achieved good results in the SAT Race competition [25]) - Eureka [22] - is not provided at all (no source code nor executables are provided).

We also tested our approach on SAT instances where the preprocessing stage does not terminate by the answer that the given SAT instance cannot be satisfied. This is the situation when the problem is not decided by the preprocessing stage and a new equivalent SAT instance is produced and sent to the solver. In such situations our method does not provide competitive results. The resulting formula is typically solved faster by the SAT solver but the preprocessing stage takes too much time. The unaffordable time consumption in the preprocessing stage is caused by extensive propagation performed by the method by which huge numbers of conflicts are inferred. It seems that on these problems the proposed approach is too strong and represents an overhead only. The numbers of inferred conflicts is not proportional to the time saved in the search for the solution stage. Moreover, as it was mentioned in the introduction, there is almost no room for improving the SAT solvers on such easy (satisfiable) SAT instances. However, this disadvantage may be overcome firstly by a better implementation of our technique (our current implementation is an experimental prototype and

the quality of our code is uncompetitive with the quality of code of the tested SAT solvers) and secondly by making the propagation less extensive on problems with many conflicts (that is, not to infer all the conflicts).

The question may now be what to do with the method at current stage of implementation when we have a new problem of unknown difficulty. That is shall we use the method or the SAT solver of our choice directly? Technically we can answer this question as follows. We can run both the preprocessing method and the SAT solver in parallel. On a machine with more than one processor we obtain an exponential speedup (the method succeeds) or no improvement. On a machine with only one processor we may obtain an exponential speedup at the expense of constant slowdown. However, the ultimate goal of our implementing efforts is to answer this question automatically within the preprocessing phase.

4 Related Works

Our method for SAT problem reformulation was originally proposed for solving planning problems using planning graphs. It was named projection consistency and it was described in the technical report [28] by Surynek. Clique-consistency proposed in this paper is an adaptation of projection consistency for the SAT domain. In addition to the description of projection consistency, the technical report contains theoretical comparison of the proposed consistency with arc-consistency and singleton arc-consistency (briefly said AC and SAC can be simulated by projection consistency; moreover there are cases on which projection consistency propagates while AC and SAC do not; the similar results hold for clique-consistency too).

The idea of exploiting structural information for solving problems is not new. There is a lot of works concerning this topic. Many of these works are dealing with methods for breaking symmetries [2, 4, 8]. We share the goal with these methods, which is to reduce the search space. However, we differ in the way how we are doing this. We are rather trying to infer what would happen if the search over the problem proceeds in some way. And if that direction seems to be unpromising the corresponding part of the search space is skipped. Symmetry breaking methods are rather trying not to do the same work twice (or more times) by a clever transformation of the original problem.

Our work was much influenced by the paper of Aloul, Markov and Sakallah [2]. We are studying the same set of difficult SAT problems. Nevertheless, it seems that our method is simpler to implement and more effective on the set of selected testing problems.

Another original approach to solving SAT problems is to exploit integer programming (IP) techniques. An interesting combination of IP and SAT techniques is given in [19]. The proposed IP approach is especially successful on difficult SAT problems.

Finally let us note that the detection of cliques in the structure of the problem is not new. A work dealing with a consistency based on cliques of inequalities was published by Sqalli and Freuder [26]. They use information about cliques to reach more global reasoning about the problem. Another work dealing with the similar ideas is [13] in which the authors use a graph structure of the problem to transform it into

another formulation based on global constraints, which provide stronger propagation than the original formulation.

5 Conclusions and Future Work

We proposed a method for preprocessing difficult (unsatisfiable) SAT instances based on the greedy clique decomposition of the transformed input CNF formula. Although the method is not universal it provides improvements in the order of magnitude compared to the state-of-the-art SAT solvers on tested SAT instances. Moreover, our method can be easily integrated into a SAT solver (new or existing) which may significantly improve its performance on difficult SAT instances.

For future we plan to further tune the method to be able to cope better with the problems having few edges in the graphs of conflicts (for example Urquhart instances). This may be done by some alternative consistency technique instead of singleton arc-consistency. We also plan to investigate the possibility to make the preprocessing iterative. That is to further preprocess the formula resulting from the previous preprocessing.

Another issue worth a deeper study is how the cliques of the clique decomposition should look like in order to our method can succeed. Our further experiments showed that better results can be obtained by using a clique decomposition where sizes of the individual cliques differ little (having several cliques of the similar size is better than having one large clique and several much smaller cliques).

We also plan to write an experimental SAT solver which would utilize the clique-consistency during search. This may be useful for early determining that a certain part of the search space does not contain a solution.

Finally an interesting research direction is some kind of a combination of existing symmetry breaking methods and the proposed clique-consistency.

References

1. Aloul, F.A.: Fadi Aloul's Home Page - SAT Benchmarks. Personal Web Page. University of Michigan, USA (March 2007) <http://www.eecs.umich.edu/faloul/benchmarks.html>
2. Aloul, F.A., Ramani, A., Markov, I.L., Sakallah, K.A.: Solving Difficult SAT Instances in the Presence of Symmetry. In: (DAC 2002). Proceedings of the 39th Design Automation Conference, USA, pp. 731–736. ACM Press, New York (2002)
3. Allen, J., Hendler, J., Tate, A. (eds.): Readings in Planning. Morgan Kaufmann, San Francisco (1990)
4. Benhamou, B., Sais, L.: Tractability through Symmetries in Propositional Calculus. *Journal of Automated Reasoning*, vol. 12-1, pp. 89–102. Springer, Heidelberg (1994)
5. Bessière, C., Debruyne, R.: Optimal and Suboptimal Singleton Arc Consistency Algorithms. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005), pp. 54–59, Canada, Professional Book Center (2005)
6. Blum, A.L., Furst, M.L.: Fast Planning through Planning Graph Analysis. In: *Artificial Intelligence 90*, pp. 281–300. AAAI Press, Stanford, California, USA (1997)

7. Cook, S.A.: The Complexity of Theorem Proving Procedures. In: Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, USA, pp. 151–158. ACM Press, NewYork (1971)
8. Crawford, J.M., Ginsberg, M.L., Luks, E.M., Roy, A.: Symmetry-Breaking Predicates for Search Problems. In: KR-96. Proceedings of the 5th International Conference on Principles of Knowledge Representation and Reasoning, pp. 148–159. Morgan Kaufmann, San Francisco (1996)
9. Dechter, R.: Constraint Processing. Morgan Kaufmann, San Francisco (2003)
10. Dowling, W., Gallier, J.: Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming* 1(3), 267–284 (1984)
11. Eén, N., Sörensson, N.: MiniSat — A SAT Solver with Conflict-Clause Minimization. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, Springer, Heidelberg (2005)
12. Eén, N., Sörensson, N.: The MiniSat Page. Research Web Page. Chalmers University, Sweden (March 2007) <http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/Main.html>
13. Frisch, A.M., Miguel, I., Walsh, T., CGRASS.: A System for Transforming Constraint Satisfaction Problems. In: O’Sullivan, B. (ed.) Recent Advances in Constraints. LNCS (LNAI), vol. 2627, pp. 15–30. Springer, Heidelberg (2003)
14. Fu, Z., Marhajan, Y., Malik, S.: zChaff. Research Web Page. Princeton University, USA, (March 2007) <http://www.princeton.edu/~chaff/zchaff.html>
15. Gershman, R., Strichman, O.: HaifaSat – a new robust SAT solver. Research Web Page. Technion Haifa, Israel, (March 2007) <http://www.cs.technion.ac.il/~gershman/HaifaSat.htm>
16. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Academic Press, London (1980)
17. Kautz, H.A., Selman, B.: Planning as Satisfiability. In: ECAI-92. Proceedings of the 10th European Conference on Artificial Intelligence, Austria, pp. 359–363. John Wiley and Sons, Chichester (1992)
18. Le Berre, D., Simon, L.: SAT Competition 2005. Competition Web Page.Scotland (March 2007) <http://www.satcompetition.org/2005/>
19. Li, R., Zhou, D., Du, D.: Satisfiability and integer programming as complementary tools. In: ASP-DAC 2004. Proceedings of the 2004 Conference on Asia South Pacific Design Automation: Electronic Design and Solution Fair 2004, Japan, pp. 879–882. IEEE Press, Orlando, Florida, USA (2004)
20. Mackworth, A.K.: Consistency in Networks of Relations. In: Artificial Intelligence, vol. 8, pp. 99–118. AAAI Press, Stanford, California, USA (1977)
21. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an Efficient SAT Solver. In: DAC-2001. Proceedings of the 38th Design Automation Conference, USA, pp. 530–535. ACM Press, NewYork (2001)
22. Nadel, A.: Alexander Nadel’s Page. Research Web Page. Tel Aviv University, Israel (March 2007) <http://www.cs.tau.ac.il/~ale1/>
23. Nam, G.-J., Sakallah, K.A., Rutenbar, R.: A New FPGA Detailed Routing Approach via Search-Based Boolean Satisfiability. In: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 21-6, pp. 674–684. IEEE Press, Orlando, Florida, USA (2002)
24. Pipatsrisawat, K., Darwiche, A.: RSat - ...veRSATile... Research Web Page. University of California Los Angeles, USA (March 2007) <http://reasoning.cs.ucla.edu/rsat/>
25. Sinz, C.: SAT-Race 2006. Competition Web Page, USA, (March 2007) <http://fmv.jku.at/sat-race-2006/>

26. Sqalli, M.H., Freuder, E.C.: Inference-Based Constraint Satisfaction Supports Explanation. In: AAAI-96 / IAAI-96. Proceedings of the 13th National Conference on Artificial Intelligence and 8th Innovative Applications of Artificial Intelligence Conference, pp. 318–325. AAAI Press / The MIT Press, Stanford/Cambridge (1996)
27. Strichman, O., Gershman, R.: HaifaSat: a New Robust SAT Solver. In: Ur, S., Bin, E., Wolfsthal, Y. (eds.) Hardware and Software, Verification and Testing. LNCS, vol. 3875, pp. 76–89. Springer, Heidelberg (2006)
28. Surynek, P.: Projection Global Consistency: An Application in AI Planning. Technical report, ITI Series, 2007-333, Charles University, Prague, Czech Republic <http://iti.mff.cuni.cz/series> (2007)
29. Urquhart, A.: Hard Examples for Resolution. In: Journal of the ACM, vol. 34, pp. 209–219. ACM Press, NewYork (1987)
30. Velev, M.N., Bryant, R.E.: Effective Use of Boolean Satisfiability Procedures in the Formal Verification of Superscalar and VLIW Microprocessors. Journal of Symbolic Computation (JSC) 35-2, 73–106 (2003)
31. Zhang, H., Stickel, M.: An efficient algorithm for unit-propagation. In: Proceedings of the 4th International Symposium on Artificial Intelligence and Mathematics, USA (1996)

Abstraction and Complexity Measures

Lorenza Saitta¹ and Jean-Daniel Zucker^{2,3}

¹ Università del Piemonte Orientale, Dipartimento di Informatica
Via Bellini 25/G, Alessandria, Italy

² LIM&BIO, EA3502, Univ. Paris 13, 74 rue Marcel Cachin, 93017 Bobigny, France

³ UR 079 GEODES, IRD, 32 avenue Henri Varagnat, 93143 Bondy, France

Abstract. Abstraction is fundamental for both human and artificial reasoning. The word denotes different activities and process, but all are intuitively related to the notion of complexity/simplicity, which is as elusive a notion as abstraction. From an analysis of the literature on abstraction and complexity it clearly appears that it is unrealistic to find definitions valid in all disciplines and for all tasks. Hence, we consider a particular model of abstraction, and try to investigate how complexity measures could be mapped to it. Preliminary results show that abstraction and complexity are not monotonically coupled notions, and that complexity may either increase or decrease with abstraction according to the definition of both and to the specificities of the considered domain.

1 Introduction

Abstraction is an essential activity in human perception and reasoning. In Artificial Intelligence it has been investigated in problem solving [25,22,9,6,13], in problem reformulation [18,31], and also in machine learning [3,20,8,16,1].

In all the disciplines where it plays an important role, abstraction is intuitively related to the notion of *complexity/simplicity*, but this link does not make its definition any easier, as complexity (or simplicity) seems to be an equally elusive notion [19,29,5]. In fact, in the last decade a great number of different definitions of complexity have been put forwards: in Complex System theory, in Physics and in Biology the notion of *complexity* has been widely debated without arriving at a received view.

Even from a superficial analysis of the literature on both abstraction and complexity it clearly appears that it is unrealistic to hope to find a definition of either abstraction or complexity valid in all disciplines and for all tasks. Nevertheless, abstraction appears to be an ideal framework to investigate system complexity, as it is related to all its important aspects, namely the *scale*, the *relevance* and the *organization degree*. As several of the introduced measures of complexity are actually uncomputable, we take a more practical approach, by considering computable approximations of them, and then trying to investigate how those measures could be possibly mapped to a specific model of abstraction. Even though the investigation is very preliminary, it may be a starting point for a fruitful cross-fertilization between works in abstraction and complexity.

Due to the difficulties of defining abstraction in general, recent work concentrated rather on abstraction as a hierarchical representation, or to its use in concrete problem solving (see, for instance, [12], [4]). Most existing formal or informal definitions of abstraction are somewhat circular: they define abstraction as a mapping from complex to simpler representations, without specifying what the intended notion of simplicity is. The idea of this paper is to start from a definition of abstraction that is not directly related to any specific notion of complexity (yet being intuitively acceptable), and, afterwards, to use this definition to investigate whether it matches some of the existing complexity measures, sampled from the different approaches to the problem, in order to see if abstraction is actually counter-varying w.r.t. complexity (*i.e.*, more abstract is less complex).

2 Theories of Abstraction

Before describing some models of abstraction proposed in the literature, we briefly recall the representation framework \mathcal{R} proposed by Saitta and Zucker [27, 28], which has the potential of unifying previously proposed approaches, and is presented in Fig. 1. The framework is based on the observation that when a concrete problem has to be solved, a conceptualization of the domain of interest is necessary. This conceptualization involves both general knowledge, consisting of time-invariant laws and statements, independent of the specific problem at hand (we call this a *theory* \mathcal{T}), and problem-specific, *contingent* knowledge, related to the specific situation. Moreover, the knowledge involved can be represented both intensionally (by means of a *language* \mathcal{L}) and extensionally (in a *database* \mathcal{D}). The language is exactly the bridge between the two representations, the syntactic and the semantic one. The model has already been used in several applications, such as machine learning [8], cartographic generalization [20], databases [30], symbol grounding in robotic vision [28], and model-based diagnosis [26].

In order to solve a problem or to perform a task, an interaction with the external world is necessary. We call the process of acquiring information from the

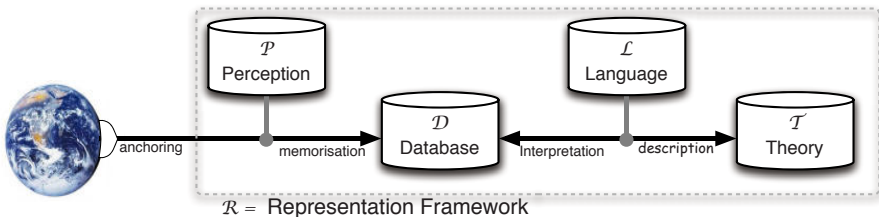


Fig. 1. *Representation Framework* \mathcal{R} . The theory \mathcal{T} contains general knowledge, expressed by means of the language \mathcal{L} . The database \mathcal{D} contains the specification of the particular individuals/objects considered in the domain, together with their attributes, and a set of relations and functions involving them.

external world a *perception* \mathcal{P} . The perception includes both a description of the specific context/problem at hand, and the type of sensors used to acquire the information. The primary role of perception is to bias the knowledge representation and reasoning processes, as Goldstone and Barsalou [10] claim it happens for human reasoning. Formally, we introduce the following:

Definition 1. A Representation Framework \mathcal{R} is a 4-ple $(\mathcal{P}, \mathcal{D}, \mathcal{L}, \mathcal{T})$, where \mathcal{P} is a perception, \mathcal{D} is a database, \mathcal{L} is a language, and \mathcal{T} is a theory.

In the past, abstraction mappings have been defined referring to single components of \mathcal{R} . In order to unify the description of these mappings, let us consider a (part of the) world \mathcal{W} to be analysed, and let \mathcal{R}_g be a representation of \mathcal{W} , which we call, conventionally, *ground*. Let \mathcal{R}_a be another representation of \mathcal{W} , which we call, instead, *abstract*. The relations between \mathcal{R}_g and \mathcal{R}_a are described in Fig. 2. For each component in \mathcal{R} we define a set of *abstraction operators*, which transform a ground representation of the appropriate type into the corresponding abstract one. In particular, operators denoted by ω operate at the perception level, operators denoted by δ operate at the database level, operators denoted by λ operate at the language level, and operators denoted by τ operate at the theory level.

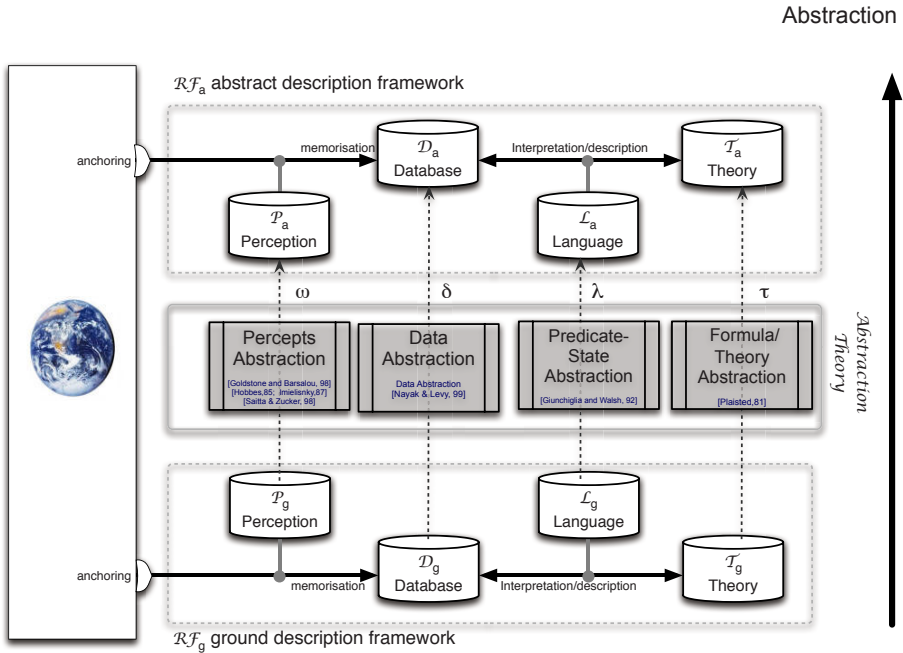


Fig. 2. Relation between a ground and an abstract representation framework. Abstraction mappings can be defined for any of the components of the framework.

Informally, an *abstraction* is a transformation from a ground framework \mathcal{R}_g to an abstract one, \mathcal{R}_a , i.e., $\mathcal{R}_a = \mathcal{A}(\mathcal{R}_g)$, where $\mathcal{A} = \{\omega, \delta, \lambda, \tau\}$, and $\mathcal{P}_a = \omega(\mathcal{P}_g)$, $\mathcal{D}_a = \delta(\mathcal{D}_g)$, $\mathcal{L}_a = \lambda(\mathcal{L}_g)$, and $\mathcal{T}_a = \tau(\mathcal{T}_g)$.

Previous theories of abstraction mostly focused on one of this operator types. For instance, Plaisted [22] has provided a foundation of theorem proving with abstraction, which is seen as a mapping from a set of clauses to another one that satisfies some properties. This approach, which corresponds to defining some operator τ at the theory level, was not concerned with any other of the representation components in Fig. 2.

Tenenberg [32] starts from Plaisted's work and defines abstraction as a mapping between predicates, which preserves logical consistency. He defines an abstraction either as (i) a predicate mapping (a renaming of predicate symbols), or (ii) a mapping of clauses based on predicate mapping, where only consistent clauses are kept. Tenenberg's definition of abstraction is therefore both at the language level (operator type λ) and at the theory level (operator type τ).

Giunchiglia and Walsh [9] have extended Plaisted's approach and reviewed most of the work done at the time in reasoning with abstraction. They informally define abstraction as a mapping from a ground representation onto an abstract one, which preserves certain desirable properties and is simpler to handle. Like in Tenenberg's approach, Giunchiglia and Walsh's abstraction is defined at the level of language (λ) and theory (τ).

On the contrary, Nayak and Levy [21] proposed a semantic theory of abstraction. This theory defines abstraction as a *model level* mapping rather than predicate mapping, i.e., abstraction is defined at the database level (operator type δ). Nayak and Levy's view of abstraction is a two-step process: the first one consists in a limited abstraction of the domain model, and the second one in constructing a set of abstract formulas to capture the abstracted domain model (operator types λ and τ).

Finally, other approaches can be implicitly or explicitly cast at the "perception" level; for instance, Hobbs [11] aimed at generating, out of an initial theory, computationally more tractable ones, by focusing on the *granularity* of objects or observations. In an analogous way, Imielinski [14] proposed an approximate reasoning framework for abstraction, by defining an *indistinguishability* relation among objects of a domain. Saitta and Zucker [27] proposed the \mathcal{KRA} model, introducing the possibility of defining abstraction also at the perception level.

The previously mentioned definitions of abstraction mappings are special cases of the \mathcal{KRA} model, which, in addition, offers the means to compare them across all representation components.

3 The \mathcal{KRA} Abstraction Model

In this section we briefly describe the \mathcal{KRA} model. Discussing the merits/drawbacks of this model is out of the scope of this paper and we take it as a given framework for analysing abstraction.

Let $\mathcal{R} = (\mathcal{P}, \mathcal{D}, \mathcal{L}, \mathcal{T})$ be a representation framework, describing a world \mathcal{W} in view of performing some task. Relevant information from \mathcal{W} is obtained via the perception \mathcal{P} , which consists of the 5-ple $\mathcal{P} = (\mathbf{OBJ}, \mathbf{ATT}, \mathbf{FUNC}, \mathbf{REL}, \mathbf{OBS})$, where **OBJ** contains the types of objects considered in \mathcal{W} , **ATT** denotes the types of attributes of the objects, **FUNC** specifies a set of functions (in particular the *sensors* used to acquire information from \mathcal{W}), and **REL** is a set of relations among object types. The set **OBS** of observations contains the actual measurements of the specific problem/task, including the actual objects considered (having one of the type in **OBJ**), the values of their attributes (which are described in **ATT**), the values of the functions (which are described in **FUNC**), and the tuples of objects satisfying the relations (which are described in **REL**).

More precisely, these sets can be expressed as follows:

- **OBJ** = $\{\mathbf{TYPE}_i | 1 \leq i \leq N\}$
- **ATT** = $\{A_j : \mathbf{TYPE}_j \rightarrow A_j | 1 \leq j \leq M\}$
- **FUNC** = $\{f_k : \mathbf{TYPE}_{i_k} \times \mathbf{TYPE}_{j_k} \times \dots \rightarrow C_k | 1 \leq k \leq S\}$
- **REL** = $\{r_h \subseteq \mathbf{TYPE}_{i_h} \times \mathbf{TYPE}_{j_h} | 1 \leq h \leq R\}$

The perception defines an ontology of the domain and grounds the theory. The set **OBS** cannot be defined in general, because it refers to a particular system. For the sake of exemplification we introduce a simple running example, that will be used throughout the paper.

Example 1. Let us consider the world consisting of pictures with $n \times m$ pixels. We can define the following perception:

- **OBJ** = **PIXEL**
- **ATT** = $\{x : \mathbf{PIXEL} \rightarrow X; y : \mathbf{PIXEL} \rightarrow Y\}$
- **FUNC** = $\{color : \mathbf{PIXEL} \rightarrow \mathbf{PALETTE}\}$
- **REL** = $\{samecolor \subseteq \mathbf{PIXEL} \times \mathbf{PIXEL}\}$

The above perception states that our world consists of pictures composed by nm pixels, each one specified by a unique identifier and by its coordinates (x, y) . A color sensor assigns to each pixel one element of a color palette. Finally, a color comparator perceives that two pixels are of the same color. For the co-domains of the functions we have: $X = \{1, 2, \dots, n\}$, $Y = \{1, 2, \dots, m\}$, and **PALETTE** = $\{white, orange, pink, \dots, black\}$ [24].

Observations **OBS** includes the identifiers $a_1, \dots, a_{n \cdot m}$ of the pixels (ordered, for instance, by rows), the values $\{x(a_i), y(a_i), color(a_i) | 1 \leq i \leq n \cdot m\}$, and the set of pairs $\{(a_i, a_j) | color(a_i) = color(a_j), 1 \leq i, j \leq n \cdot m\}$.

Concerning the language, we choose a typed logical language $\mathcal{L} = \{\mathbf{P}, \mathbf{F}, \mathbf{C}\}$, with the set of predicates $\mathbf{P} = \{\mathbf{pixel}(v), \mathbf{x}(v, i), \mathbf{y}(v, j), \mathbf{color}(v, \lambda_k), \mathbf{adj}(u, v), \mathbf{samecolor}(u, v), \mathbf{equal}(h, k)\}$, and the set of functions $\mathbf{F} = \{color : \mathbf{PIXEL} \rightarrow \mathbf{PALETTE}, |h - k| : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}, max(h, k) : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}\}$. The functions $|h - k|$ and $max(h, k)$, as well as the set \mathbb{N} of nonnegative integers, have been introduced because they are used in the theory.

The set of constants \mathbf{C} coincides with the set of objects defined in \mathbf{OBS} , because we assume that the constants denoting the objects/values in \mathcal{L} coincide with their identifiers in the domain (for instance, pixel a_i has name a_i). The theory consists of one formula, describing the adjacency of two pixels:

$$\mathcal{T} = \{ \mathbf{adj}(u, v) \leftarrow \mathbf{pixel}(u) \wedge \mathbf{pixel}(v) \wedge \mathbf{x}(u, i_1) \wedge \mathbf{y}(u, j_1) \wedge \mathbf{x}(v, i_2) \wedge \mathbf{y}(v, j_2) \wedge \mathbf{equal}(\max(|i_2 - i_1|, |j_2 - j_1|), 1) \} \quad (1)$$

Finally, we must define the database \mathcal{D} . Tables in \mathcal{D} are associated to predicates, functions and formulas in the perception and in the theory. Some tables can be derived from others using the operators of relational algebra. What to store explicitly and what to leave to be derived on demand is up to the designer. In the case of the example, we have considered the following tables:

$$\mathcal{D} = \{ \mathbf{TablePixel}, \mathbf{TableColor}, \mathbf{TableSameColor}, \mathbf{TableAdjacent} \},$$

where $\mathbf{TablePixel} = (\mathbf{pixel}, x, y)$, $\mathbf{TableColor} = (\mathbf{pixel}, \mathbf{color})$, $\mathbf{TableSameColor} = (\mathbf{pixel}_1, \mathbf{pixel}_2, \mathbf{color})$, $\mathbf{TableAdjacent}(\mathbf{pixel}_1, \mathbf{pixel}_2)$.

Both $\mathbf{TablePixel}$ and $\mathbf{TableColor}$ are populated directly by \mathbf{OBS} , by measuring the coordinates of the pixels and registering their color. Using the operators of relational algebra, $\mathbf{TableSameColor}$ can be derived from $\mathbf{TableColor}$ through renaming, selection and projection operations.

Let us consider now two representation frameworks, $\mathcal{R}_g = (\mathcal{P}_g, \mathcal{D}_g, \mathcal{L}_g, \mathcal{T}_g)$ and $\mathcal{R}_a = (\mathcal{P}_a, \mathcal{D}_a, \mathcal{L}_a, \mathcal{T}_a)$. We call \mathcal{R}_g a *ground* framework. An abstraction mapping can be defined between \mathcal{R}_g and \mathcal{R}_a . Once \mathcal{P}_g is given, any particular system (in the chosen domain) can be obtained by assigning values to a set of variables (representing objects, attributes, function values, and tuples in relations). Each assignment to all these variables is a *configuration* γ_g . Let Γ_g be the set of the possible ground configurations. In an analogous way, we can define the set Γ_a of all possible abstract configurations (for more details, see [27]).

Then, we define an abstraction as follows:

Definition 2. *Given two representation frameworks \mathcal{R}_g and \mathcal{R}_a , if there exists a mapping $\Gamma_g \rightarrow \Gamma_a$, such that the mapping associates subsets of Γ_g to single elements of Γ_a , we say that \mathcal{R}_a is more abstract than \mathcal{R}_g .*

In Definition 2 abstraction is seen as a *relative* rather than an absolute notion, and is defined between representation frameworks and not single objects. The analysis of the meaning and implications of this definition is out of the scope of this paper, and we will rely on its intuitive understanding. Substantially, the definition says that abstraction performs some kind of information aggregation, simplifying thus representations, and, possibly, reasoning.

Concretely, we only consider mappings that can be obtained from the application of a set of the above introduced *abstraction operators*, which generate abstract configurations starting from ground ones. Some operators can be domain-independent, and some are domain-specific. Considering abstraction as a process consisting of operators application has the limit that not all possible mappings between \mathcal{R}_g and \mathcal{R}_a can be actually realized, but it has the advantage

that *compatibility conditions* can be requested, and quantities to be *conserved* can be specified (for instance, diagnosability). Finally, abstraction is a *transitive* relationship, and it induces a *partial order* among representation frameworks. The introduction of operators provides a constructive semantics to abstraction.

The basic idea underlying this notion of abstraction tries to capture the variety of a world, i.e., the number of possible alternative configurations. In fact, a world with just one configuration would not be interesting at all, as nothing can happen in it. On the other hand, less configurations means a greater ease to handle the world. Then, abstraction makes a world easier to be reasoned about/acted upon, but poorer in diversity.

In order to illustrate the idea, let us consider again the running example. If we look at pictures with a less sensitive color detector, we may have the color of each pixel categorized, for instance, into 8 classes, forming an abstract palette:

$$\mathbf{PALETTE}_a = \{black, violet, blue, brown, green, red, yellow, white\}.$$

In the abstract perception \mathcal{P}_a the only changes, w.r.t. \mathcal{P}_g , are the function set \mathbf{FUNC}_a , which now contains a function $color_a : \mathbf{PIXEL} \rightarrow \mathbf{PALETTE}_a$ instead of $color_g : \mathbf{PIXEL} \rightarrow \mathbf{PALETTE}_g$, and the observation \mathbf{OBS}_a , where the color of each pixel belongs now to $\mathbf{PALETTE}_a$ and not to $\mathbf{PALETTE}_g$.

The change in the perception is performed by an operator

$$\omega_{col}(\mathbf{PALETTE}_g, \mathbf{PALETTE}_a),$$

which reduces the codomain $\mathbf{PALETTE}_g$ of the function $color_g(v)$ to $\mathbf{PALETTE}_a$, by letting some of the elements of $\mathbf{PALETTE}_g$ collapse into single elements of $\mathbf{PALETTE}_a$.

In the database, the values belonging to $\mathbf{PALETTE}_g$ occur in the column "color" of both `TableColor` and `TableSameColor`. The operator

$$\delta_{col}(T, color, \mathbf{PALETTE}_g, \mathbf{PALETTE}_a),$$

corresponding to ω_{col} , changes the values of the color stored in column "color" of table `T` according to the mapping specified by ω_{col} . Operator δ_{col} is applied to `TableColor` and `TableSameColor`; the other tables remain unchanged.

Regarding the language \mathcal{L}_a , it is easy to see that $\mathbf{P}_a = \mathbf{P}_g, \mathbf{F}_a = \mathbf{F}_g - \{color_g(v)\} \cup \{color_a(v)\}, \mathbf{C}_a = \mathbf{C}_g - \mathbf{PALETTE}_g \cup \mathbf{PALETTE}_a$.

The abstract theory remains the same, namely $\mathcal{T}_a = \mathcal{T}_g$, because the single formula in the theory does not depend on the color of the pixels.

4 Complexity Measures

The study of complex systems has raised a growing number of issues related to the understanding of the very nature of complexity. Although there is no general agreement on the definition of complexity, there is a convergence on two basic idea: the first is that structure, organization, and diversity are key features of complex systems, whereas the second is that complexity should not

monotonically increase with disorder. In this section we mention a few of the measures proposed in the literature, trying to link them to the notions introduced in Section 3. The complexity measures have been sampled among the existing ones, and are meant to be representative of different approaches to describe complexity. In particular, we will consider measures based on Turing machines, and statistical measures either related to order/disorder or to self-similarity.

4.1 Turing Machine-Based Complexity Measures

One of the first and better known measure of complexity is *Kolmogorov-Chaitin Algorithmic Complexity*, also called *Algorithmic Information Content (AIC)*, which states that the complexity of an object x is the length, in bits, of the shortest program that, run on a Universal Turing Machine, outputs x and halts. Then, given a universal Turing machine \mathcal{U} , we can write:

$$\mathcal{K}(x) = \min_{\pi | \mathcal{U}(\pi)=x} \ell(\pi) + \mathcal{O}(1), \quad (2)$$

where π is a program on \mathcal{U} and $\ell(\pi)$ is its length in bits. The complexity $\mathcal{K}(x)$ is machine-independent up to an additive constant. Unfortunately, Kolmogorov complexity is not computable.

Starting from *AIC*, Vitanyi [33] observed that the information provided by this measure can be divided into two parts: one accounting for the useful regularities present in the object, i.e., the *meaningful* information, and one accounting for the remaining *accidental* information. According to Vitanyi, it is the first one which is most useful. An idea similar to Vitanyi's has been proposed both by Gell-Mann and Lloyd [7], who introduced the *effective complexity* (the length of a highly compressed description of the regularities of an object), and by Koppel [17], who called *sophistication* the useful part of the two-part code of the description encoding.

All the above measures capture a complexity linked to the *description* of an object, but tell nothing about how difficult it is to actually (re)construct the object. Criticizing Kolmogorov's approach, Bennett [2] invokes, as a better notion of an object's complexity, the time required by a Turing machine to actually generate the object, which he call *logical depth*. The notion of complexity defined by Bennett may be at odds with Kolmogorov's one, as it is possible that an increase in one measure corresponds to a decrease in the other. To complete his definition, Bennett chooses, for computing the complexity of an object, a program whose length is no more than s bits longer than the shortest one. It is clear that the logical depth (or approximations thereof) also depends on a Turing Machine, and, hence, cannot be computed.

4.2 Statistical Complexity Measures

Statistical measures of complexity tend to link complexity to disorder/order, and usually they make use of the entropy of a system. Starting from the observation

that both totally ordered systems and totally disordered ones are intuitively simple, Lopez-Ruiz et al. [23] have proposed a complexity measure, called *normalized complexity*, which is low in both those cases and increases in between. This measure takes into account both the information stored in a system (its *entropy* H), and its *disequilibrium* D , i.e., its distance from an equiprobable distribution (N is the number of states):

$$D = \sum_{i=1}^N \left(p_i - \frac{1}{N} \right)^2 \tag{3}$$

The normalized complexity is then defined as $\mathcal{C}_{LMC} = H \cdot D$.

Notwithstanding its intuitive appeal, this measure has been considered by Feldman and Crutchfield [5] as inadequate to capture complexity, because the disequilibrium term is a non extensive¹ quantity. In order to overcome this drawback, these authors propose a modification to \mathcal{C}_{LMC} : they express D by means of the Kullback-Leibler distance, obtaining thus:

$$\mathcal{C}'(Y) = H(Y) \cdot KL(Pr(y)||1/N) \tag{4}$$

In an attempt to unify a number of previously proposed complexity measures, Shiner et al. [15] have proposed a parameterized measure, namely a *simple measure* of complexity $\Gamma^{(\alpha\beta)}$, which, by varying its two parameters α and β , shows different types of behavior: increasing with *order*, increasing with *disorder*, or reaching a maximum between order and disorder. The measure $\Gamma^{(\alpha\beta)}$ is easy to compute and is independent from the system size. By defining the *disorder* as $\Delta = H/H_{max}$, where H is the system entropy, and the *order* as $\Omega = 1 - \Delta$, $\Gamma^{(\alpha\beta)}$ is defined as:

$$\Gamma^{(\alpha\beta)} = \Delta^\alpha \Omega^\beta \tag{5}$$

$\Gamma_g^{(\alpha\beta)}$, as a function of Δ , is 0 at the extremes of the interval $[0, 1]$ and has a maximum at $\Delta = \frac{\alpha}{\alpha+\beta}$.

A rather different approach is taken by Wolpert and McCreedy [34], who base their notion of complexity on the *self-dissimilarity* degree a system exhibits when analysed at different scales. The idea is to consider an embedded sequence of spaces (scales) Ω_s . At a given scale, let π_s be the probability distribution over the system's states. If two scales $s_2 > s_1$ are considered, the mapping between the two is described by a set of mapping $\rho_{s_1 \leftarrow s_2}^{(i)}$. The knowledge of π_{s_2} and $\rho_{s_1 \leftarrow s_2}^{(i)}$ allows the new distribution π_{s_1} to be computed. In order to make a comparison between the scales, these last are reduced to a single one by means of the transformations $s_2 \rightarrow s_c$ and $s_1 \rightarrow s_c$, where $s_c \geq \text{Max}[s_1, s_2]$. When the transformation is accomplished, the two resulting distributions can be compared, using, for instance, the Kullback-Leibler distance.

¹ An extensive quantity diverges as $\mathcal{O}(N)$ in the thermodynamical limit, i.e. when the number of states $N \rightarrow \infty$.

5 Abstraction and Complexity

Given an abstraction theory relating two frameworks, namely $\mathcal{R}_a = \mathcal{A}(\mathcal{R}_g)$, let $\Gamma_g \rightarrow \Gamma_a$ be the abstraction mapping which it is based upon. In this section we consider the relations between the abstraction defined in Section 3 and the complexity measures mentioned in Section 4. The analysis will be mostly qualitative in nature, but sufficient to point out that the relation between the two notions is far from obvious.

5.1 Turing Machine-Based Complexity Measures

Let us consider first Kolmogorov complexity $\mathcal{K}(\gamma)$. Even though $\mathcal{K}(\gamma)$ is uncomputable, we can use an approximation of it by choosing a specific encoding for the configurations in our domain. In fact, what we are interested in is not to find the optimal (shortest) description of objects, but to see how a given encoding changes across abstraction levels.

A very basic (but also general) encoding, independent of any consideration about the type and frequency of the actual configurations $\gamma \in \Gamma$, is to establish a one-to-one association between the elements of Γ and the integers between 1 and $N = |\Gamma|$ (codes). In this way, we can describe/transmit a particular γ simply transmitting the integer that is its code. In this case, the cost in bits of the description (its Kolmogorov complexity) is $\mathcal{K}(\gamma) = \lg_2 N$ (ignoring the constant). This value is the maximum needed to transmit ANY configuration. If the type of the configurations is specified, other, more pertinent codings can be used, of course. If we consider the two sets Γ_g and Γ_a , we obtain: $\mathcal{K}(\gamma_a) = \lg_2 N_a < \lg_2 N_g = \mathcal{K}(\gamma_g)$. In fact, we have $N_a < N_g$ by definition. Then, Kolmogorov complexity decreases when abstraction increases.

The same is true for Vitanyi's *meaningful information*, Gell-Mann and Lloyd's *effective complexity*, and Koppel's *sophistication*. In fact, the first part of a two-part code for Γ is actually a program on a Turing machine, which describes the regularities shared by many elements of it, leaving out exceptions (configurations that the model does not match). Then, if M_1 is the number of possible models (with a given a-priori form) and N_2 the number of exceptions, the complexity of the above introduced coding is at most $\mathcal{C}(\Gamma) = \lg_2 M_1 + \lg_2 \binom{N}{N_2}$. Moving from Γ_g to Γ_a , existing regularities are conserved, so that the number of possible models either decreases or remains constant. On the other hand, some configurations that were exceptions in the ground space may be described by the abstract model, so that their number either decreases or remains constant, as well, without increasing M_1 .

For the sake of illustration, let us consider our running example. We have $N_g = S_g^{nm}$, with $S_g = |\mathbf{PALETTE}_g|$, and $N_a = S_a^{nm}$, with $S_a = |\mathbf{PALETTE}_a|$. A configuration corresponds to a particular picture. Given any picture γ_g in the ground space, it can be described with $\mathcal{K}(\gamma_g) = n \cdot m \cdot \lg_2 S_g$ bits. If a user receives this number, she is able to reconstruct the image, because she knows

which one it is. The abstract picture γ_a requires only $\mathcal{K}(\gamma_a) = n \cdot m \cdot \lg_2 S_a$ bits to be described. The complexity so evaluated is actually the one for printing the picture, and is independent of the actual content (for instance, a uniformly red picture and one with a complicated spatial distribution of colors require the same number of bits).

Concerning the two-part code for describing the pictures, let us suppose that we only consider ground pictures of sea, with rows m_1 through m_2 containing only blue pixels (water), rows $(m_2 + 1)$ through m containing only hazel pixels (sky), whereas the other rows may contain pixels of any colors. A model consists of the pair (m_1, m_2) . Then $M_{1,g} = \frac{m^2 - 3m + 2}{2}$. If the operator ω_{col} is applied, the values "blue" and "hazel" collapse in the abstract space, and the model set reduces to the pictures with the two upper strips collapsing. The number of model is then $M_{1,a} = m - 1$, which is smaller than $M_{1,g}$ for $m \geq 4$. At the same time, some rows that had both blue and hazel pixels now became uniform, and the corresponding picture, that was an exception in the ground space, is now covered by the abstract model, reducing thus the number of exceptions.

The analysis of Bennett's *logical depth* is more difficult. For simplicity, let us assume that, given a configuration γ , its logical depth be the run time taken by the program π , whose length is exactly its Kolmogorov complexity. Without specifying the nature of the configurations, it is not possible to say whether π_a will run faster than π_g . In fact, even though the abstract configurations are "simpler" (according to Kolmogorov) than the ground ones, they may be more difficult to generate. Then, the evaluation of the logical depth depends upon the content of the considered world, and nothing can be said in general, except that an abstract framework can be, according to Bennett, either simpler or more complex than the ground one. This results derives from the fact that the logical depth is a generative and not a descriptive measure of complexity.

5.2 Stochastic Measures of Complexity

In order to consider stochastic approaches to complexity, we need to extend the representation framework described in Section 3 in order to accomodate probability. In particular, let us define a *Stochastic Representation Framework* \mathcal{SR} as a pair (\mathcal{R}, π) , where \mathcal{R} is a representation framework, and π is a probability distribution over the set Γ of configurations.

In order to compute the complexity in the abstract space, we have also to extend the abstraction mapping:

Definition 3. *Given two stochastic representation framework $\mathcal{SR}_g = (\mathcal{R}_g, \pi_g)$ and $\mathcal{SR}_a = (\mathcal{R}_a, \pi_a)$, we require that*

$$\pi_a(\gamma_a) = \sum_{\gamma_g \in G_g(\gamma_a)} \pi_g(\gamma_g) \quad \text{for each } \gamma_a,$$

where $G_g(\gamma_a)$ is the set of ground configurations that have γ_a as their image.

Let us consider now Lopez-Ruiz et al.'s *normalized complexity* \mathcal{C}_{LMC} , which is the product of the entropy and the diversity of Γ . Then, the normalized complexity of the ground space is:

$$\begin{aligned} \mathcal{C}_{LMC}(\Gamma_g) &= H(\Gamma_g) \cdot D(\pi_g) = \\ &= - \sum_{\gamma_g \in \Gamma_g} \pi_g(\gamma_g) \log_2 \pi_g(\gamma_g) \cdot \sum_{\gamma_g \in \Gamma_g} \left(\pi_g(\gamma_g) - \frac{1}{N_g} \right)^2 \end{aligned}$$

In the abstract space, we have instead:

$$\begin{aligned} \mathcal{C}_{LMC}(\Gamma_a) &= H(\Gamma_a) \cdot D(\pi_a) = \\ &= - \sum_{\gamma_a \in \Gamma_a} \pi_a(\gamma_a) \log_2 \pi_a(\gamma_a) \cdot \sum_{\gamma_a \in \Gamma_a} \left(\pi_a(\gamma_a) - \frac{1}{N_a} \right)^2 \end{aligned}$$

Exploiting the properties of the entropy function, we get $H(\Gamma_a) \leq H(\Gamma_g)$. For what concerns the disequilibrium factor, we obtain $D(\Gamma_a) \leq D(\Gamma_g)$ when:

$$\sum_{\gamma_a \in \Gamma_a} \sum_{\gamma_1 \in S_a} \sum_{\gamma_2 \in S_a, \gamma_2 \neq \gamma_1} \pi_g(\gamma_1) \pi_g(\gamma_2) \leq \frac{1}{N_a} - \frac{1}{N_g} \tag{6}$$

Equation (6) may or may not be verified. As an example, if the probability distribution over the ground configurations is uniform, then $D(\Gamma_g) = 0$, whereas most likely $D(\Gamma_a) \neq 0$; on the contrary, a uniform distribution over the abstract configurations may not derive from a uniform distribution over Γ_g . Then, there is no fixed relation between $D(\Gamma_g)$ and $D(\Gamma_a)$, and hence, $\mathcal{C}_{LMC}(\Gamma_a) \geq \mathcal{C}_{LMC}(\Gamma_g)$.

Using our running example, let us assume that, for each pixel, all colors in **PALETTE**_g are equally likely, i.e., $Pr(color_g(v) = \lambda_k) = \frac{1}{S_g}$ for each $v \in \mathbf{PIXEL}$; moreover, colors are assigned to the pixels independently. Then, each configuration γ_g has the same probability $\frac{1}{N_g} = S_g^{-nm}$, because each of the nm pixel has the same probability to be of any of the colors. In this case, the entropy $H(\Gamma_g) = nm \log_2 S_g$ and $D(\Gamma_g) = 0$; then, $\mathcal{C}_{LMC}(\Gamma_g) = 0$.

Let us associate now, in **PALETTE**_a, 2 ground colors to *violet*, 3 ground colors to *brown*, 4 ground colors to *green* and *red*, and 5 ground colors to *blue* and *yellow*, respectively, and keep *black* and *white* unchanged. The abstract configurations, then, are no more equally likely. More precisely, the probability distribution over abstract configurations only depends on the numbers of pixels with a given color and not on their spatial distribution on the image. Then, the probability distribution over Γ_a is a multinomial one:

$$\pi_a(\gamma_a) = Pr(n_1, \dots, n_{S_a}) = \binom{nm}{n_1 \dots n_{S_a}} \prod_{k=1}^{S_a} p_k^{n_k}, \tag{7}$$

where p_k is the sum of the probabilities, in the ground space, associated to the colors collapsed onto λ_k . Then, for Γ_a we have:

$$H(\Gamma_a) = - \sum_{\gamma_a \in \Gamma_a} \pi_a(\gamma_a) \log_2 \pi_a(\gamma_a)$$

and

$$D(\Gamma_a) = \sum_{\gamma_a \in \Gamma_a} \left(\pi_a(\gamma_a) - \frac{1}{S_a^{nm}} \right)^2$$

Finally:

$$C_{LMC}(\Gamma_a) = - \left[\sum_{\gamma_a \in \Gamma_a} \pi_a(\gamma_a) \log_2 \pi_a(\gamma_a) \right] \left[\sum_{\gamma_a \in \Gamma_a} \left(\pi_a(\gamma_a) - \frac{1}{S_a^{nm}} \right)^2 \right] > 0.$$

In this case, the normalized complexity increased with abstraction. If we want that the opposite happens, we have to consider abstractions that tend to assign probabilities to the abstract configurations as uniformly as possible.

If we consider the modification to the normalized complexity suggested by Feldman and Crutchfield (see Section 4.2), the entropy in the new measure \mathcal{C}' does not changes, whereas the disequilibrium factor is replaced by the Kullback-Leibler distance between the actual probability distribution over the configurations and the uniform one. Then, in the ground space we have:

$$\mathcal{C}'(\Gamma_g) = H(\Gamma_g) \text{KL}(\pi_g(\gamma_g) \| 1/N_g)$$

whereas in the abstract one:

$$\mathcal{C}'(\Gamma_a) = H(\Gamma_a) \text{KL}(\pi_a(\gamma_a) \| 1/N_g)$$

Again, as either one of the distributions $\pi_g(\gamma_g)$ and $\pi_a(\gamma_a)$ can be closer to the uniform one, there is no a fixed order between $\mathcal{C}'(\Gamma_g)$ and $\mathcal{C}'(\Gamma_a)$.

Let us consider Shiner et al.'s *simple complexity*. This measure is linked to the notion of order/disorder of the system under consideration. Given Γ_g , its maximum entropy is reached when all configurations have the same probabilities, i.e., $H_{g,max}(\Gamma_g) = \log_2 N_g$. The actual entropy of Γ_g is given by $H(\Gamma_g) = - \sum_{\gamma_g \in \Gamma_g} \pi_g(\Gamma_g) \log_2 \pi_g(\Gamma_g)$. Then:

$$\Delta(\Gamma_g) = \frac{H(\Gamma_g)}{H_{g,max}}$$

and

$$\Omega(\Gamma_g) = 1 - \Delta(\Gamma_g)$$

The simple complexity $\Gamma^{\alpha\beta}$ assumes then the expression:

$$\Gamma^{\alpha\beta}(\Gamma_g) = \left(\frac{H(\Gamma_g)}{H_{g,max}} \right)^\alpha \left(1 - \frac{H(\Gamma_g)}{H_{g,max}} \right)^\beta$$

The function $\Gamma^{\alpha\beta}(\Gamma_g)$ assume its maximum value when:

$$H(\Gamma_g) = \frac{\alpha}{\alpha + \beta} H_{g,max}$$

In the abstract space we will have:

$$\Gamma^{\alpha\beta}(\Gamma_a) = \left(\frac{H(\Gamma_a)}{H_{a,max}}\right)^\alpha \left(1 - \frac{H(\Gamma_a)}{H_{a,max}}\right)^\beta$$

The abstract simple complexity is lower than the ground one iff:

$$\left(\frac{H(\Gamma_a)}{H_{a,max}}\right)^\alpha \left(1 - \frac{H(\Gamma_a)}{H_{a,max}}\right)^\beta \leq \left(\frac{H(\Gamma_g)}{H_{g,max}}\right)^\alpha \left(1 - \frac{H(\Gamma_g)}{H_{g,max}}\right)^\beta$$

In other word:

$$\left(\frac{H(\Gamma_a)}{H(\Gamma_g)}\right)^\alpha \left(\frac{H_{a,max} - H(\Gamma_a)}{H_{g,max} - H(\Gamma_g)}\right)^\beta \leq \left(\frac{\log_2 N_a}{\log_2 N_g}\right)^{\alpha+\beta} \tag{8}$$

Depending on the values of α and β and on the degree of uniformity of the probability distributions over the ground and the abstract configurations, the simple complexity may be larger in any of the two spaces.

As Wolpert and Mcready’s self-dissimilarity measure is based on the notion of probability distribution difference across scales, this measure is not guaranteed either to co-vary or counter-vary with abstraction.

6 Conclusions

In this paper we have started a preliminary investigation of the possible relations between abstraction and complexity, suggested by the commonly accepted view that abstraction, considered as a representation change, reduces complexity. To this aim we have considered a particular model of abstraction, namely the \mathcal{KRA} model, which has the advantage of generalizing over several others.

Even from this limited study, it clearly emerges that the relations between abstraction and complexity are articulated and far from obvious. In fact, according to the complexity measure selected and the characteristics of the considered representation space, complexity in the abstract space may increase, decrease or stay the same, especially when probabilistic considerations are introduced. This conclusion holds for all the statistical measures of complexity considered. Only (an approximation of) Kolmogorov complexity decreases with abstraction.

This result suggests that the common understanding that abstraction reduces complexity must be take with care, because it is necessary to explicitly define what notion of complexity is taken into consideration. Moreover, this same result suggests that if specific requirements about the conservation of properties across abstraction layers are requested, they can help selecting an appropriate complexity measure, reflecting the specificity of the problem at hand. On the opposite, choosing a complexity measure appropriate to a given problem may act as powerful heuristic to limit the number of abstraction theories that are possible in principle, if we want that complexity and abstraction counter-vary.

References

1. Anderson, S., Revesz, P.Z.: Verifying the incorrectness of programs and automata. In: Zucker, J.-D., Saitta, L. (eds.) SARA 2005. LNCS (LNAI), vol. 3607, pp. 1–13. Springer, Heidelberg (2005)
2. Bennett, C.: Logical depth and physical complexity. In: Bennett, C. (ed.) *The Universal Turing Machine: A Half-Century Survey*, pp. 227–257 (1988)
3. Bredeche, N., Shi, Z., Zucker, J.-D.: Perceptual learning and abstraction in machine learning: an application to autonomous robotics. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 36(2), 172–181 (2006)
4. Choueiry, B.Y., Iwasaki, Y., McIlraith, S.: Towards a practical theory of reformulation for reasoning about physical systems. *Artificial Intelligence* 162(1-2), 145–204 (2006)
5. Feldman, D.P., Crutchfield, J.P.: Measures of statistical complexity: Why? *Physics Letters A* 238, 244–252 (1998)
6. Ellman, T.: Synthesis of abstraction hierarchies for constraint satisfaction by clustering approximatively equivalent objects. In: *International Conference on Machine Learning*, Amherst, MA, Morgan Kaufmann, Seattle, Washington, USA (1993)
7. Gell-Mann, M., Lloyd, S.: Information measures, effective complexity, and total information. *Complexity* 2(1), 44–52 (1996)
8. Giordana, A., Saitta, L.: Abstraction: a general framework for learning. In: *Working notes of the AAAI Workshop on Automated Generation of Approximations and Abstraction*, pp. 245–256, Boston, MA (1990)
9. Giunchiglia, F., Walsh, T.: A theory of abstraction. *Artificial Intelligence* 56(2-3), 323–390 (1992)
10. Goldstone, R., Barsalou, L.: Reuniting perception and conception. *Cognition* 65, 231–262 (1998)
11. Hobbs, J.: Granularity. In: *Int. Joint Conf. on Artificial Intelligence*, 432–435 (1985)
12. Holte, R.C., Grajkowski, J., Tanner, B.: Hierarchical heuristic search revisited. In: SARA, pp. 121–133 (2005)
13. Holte, R.C., Mkadmi, T., Zimmer, R.M., MacDonald, A.J.: Speeding up problem-solving by abstraction: A graph-oriented approach. *Artificial Intelligence* 85, 321–361 (1996)
14. Imielinski, T.: Domain abstraction and limited reasoning. In: *Proceedings of the Intern. Joint Conf. on Artificial Intelligence*, pp. 997–1003 (1987)
15. Shiner, J.S., Davison, M., Landsberg, P.T.: Simple measure for complexity. *Physical review E* 59, 1459–1464 (1999)
16. Knoblock, C.: Learning hierarchies of abstraction spaces. In: *6th International Workshop on Machine Learning*, pp. 241–245, Ithaca, NY (1989)
17. Koppel, M.: Complexity, depth and sophistication. *Complex Systems* 1, 1087–1091 (1987)
18. Lowry, M.: The abstraction/implementation model of problem reformulation. In: *Int. Joint Conf. on Artificial Intelligence*, pp. 1004–1010, Milano, Italy (1987)
19. Marczyk, J., Deshpande, B.: Measuring and tracking complexity in science. In: *Proceedings of the 6th International Conference on Complex Systems* (2006)
20. Mustiere, S., Zucker, J.-D., Saitta, L.: An abstraction-based machine learning approach to cartographic generalization. In: *Spatial Data Handling 2000 (SDH)*, pp. 150–163, Beijing, China (2000)
21. Nayak, P., Levy, A.: A semantic theory of abstraction. In: *International Joint Conference on Artificial Intelligence (IJCAI-95)*, pp. 192–196 (1995)

22. Plaisted, D.: Theorem proving with abstraction. *Artificial Intelligence* 16, 47–108 (1981)
23. Lopez-Ruiz, R., Mancini, H., Calbet, X.: A statistical measure of complexity. *Physics Letters A* 209, 209–321 (1995)
24. Ravishankar, K.C., Prasad, B.G., Gupta, S.K., Biswas, K.K.: Dominant color region based indexing for cbir. In: *iciap*, 00:887 (1999)
25. Sacerdoti, E.: Planning in a hierarchy of abstraction spaces. *Artificial Intelligence* 5, 115–135 (1974)
26. Saitta, L., Torasso, P., Torta, G.: Formalizing the abstraction process in model-based diagnosis. Tr cs-2006/34, Univ. of Torino, Italy (2006)
27. Saitta, L., Zucker, J.-D.: Semantic abstraction for concept representation and learning. In: *Proc. of the Intern. Symposium on Approximation, Reformulation and Abstraction (Asilomar, CA)* (1998)
28. Saitta, L., Zucker, J.-D.: A model of abstraction in visual perception. *Applied Artificial Intelligence* 15(8), 761–776 (2001)
29. Shalizi, C.: Methods and techniques of complex systems science: An overview. In: *Complex Systems Science in Biomedicine*, pp. 33–114. Springer, NewYork (2006)
30. Sheeren, D., Mustiere, S., Zucker, J.-D.: Consistency assessment between multiple representations of geographical databases: a specification-based approach. In: Fisher, P. (ed.) *Developments in Spatial Data Handling*, pp. 617–628. Springer, Heidelberg (2004)
31. Subramanian, D.: A theory of justified reformulations. In: Paul, D. (ed.) *Change of Representation and Inductive Bias*, pp. 147–167. Kluwer Academic Publishers, Boston (1990)
32. Tenenbreg, J.: Preserving consistency across abstraction mappings. In: *Proceedings of IJCAI-87*, pp. 1011–1014, Milan, Italy (1987)
33. Vitanyi, P.: Meaningful information. *IEEE Transactions on Information Theory* 52, 4617–4630 (2006)
34. Wolpert, D., Macready, W.: Self-dissimilarity: An empirically observable complexity measure. In: *Proc. of the Intern. Conf. on Complex Systems (Nashua, NH)* (1997)

Abstraction, Emergence, and Thought

Russ Abbott

Department of Computer Science, California State University,
Los Angeles, Ca.

Russ.Abbott@GMail.com

Abstract. My research focuses on the relationships among abstraction, emergence (as in the study of complex systems), and the externalization of thought as software—in particular, on the application of computer science perspectives, especially abstraction, to long-standing philosophical issues.

Keywords: Abstraction, emergence, thought externalization.

1 Emergence

A central issue in the study of complex systems is the nature of emergence, macro-level effects from micro-level causes. The fundamental insight of [1] is that phenomena that we tend to refer to as emergent are those that can be described independently of their implementation, i.e., those that implement a level of abstraction. Although levels of abstraction are common fare in computer science—and in some sense in engineering, where they are known as requirements—we tend to be surprised when we find that nature has implemented one without our help. Standard examples include *flocks* of birds. We have focussed on such examples because we have discovered how to create (something like) a flock abstraction with very simple implementation mechanisms.

Even though emergence is the implementation of a level of abstraction, interactions that occur according to the operations defined at a physically realized level of abstraction are epiphenomenal—just as interactions in software are also epiphenomenal. Both are implemented in terms of lower level operations. Thus one is faced with the confusing situation in which an abstraction is real but the actual interactions that realize operations defined by the abstraction are epiphenomenal. This has led to what I refer to as the blind spot of reductionism—the discarding of the baby of the abstraction with the bathwater of the epiphenomenal interactions.

A second paper (in preparation) discusses the nature of physical entities. We define a physical entity to be an instances of a level of abstraction. There are two kinds of physical entities. Static entities such as atoms and molecules exist in energy wells and have less mass than the aggregate mass of their components considered separately. Dynamic entities such as hurricanes, biological organisms, and social and political entities are famously far from equilibrium and depend on the importation of energy from their environment to persist. Because of their persistent structure, they exhibit reduced entropy. Hence physical entities so defined are objectively real.

Because they can be understood in isolation, static entities may be analyzed in terms of their components. Supervenience is useful. Because they are necessarily open to their environment, dynamic entities cannot be so analyzed. A flock (or a social club or a biological organism) persists even though the elements that compose it may change. For these entities supervenience is not useful. It is their structure and process that persist—maintained by energy acquired from their environment. It is the task of the special sciences, i.e., those other than fundamental physics, to identify and characterize the abstractions that nature implements.

2 Thought Externalization

Participants in the Conference on Unconventional Computation typically ask what sorts of computations can be done using other than Von Neumann computers. In [2], which won the conference best presentation award, I argued that one must first understand what we mean by *computation*. My answer is that processes in the physical world to which we apply the term *computation* are all externalized thought. Thus the job of computer science is to find ways to express our thoughts in languages so that when those expressions are performed by physical devices, the results are recognized as a reasonable approximation to our original thoughts—or perhaps clarified variants thereof. As such, computer science has rightfully been referred to as applied philosophy, a field whose job is to clarify our thoughts sufficiently so that they may be expressed in an executable language. We have done an admirable job of creating abstractions that are both executable and meaningful to us. The conceptual model of every software library or application represents such a thought clarification process. Similarly, most of the important sub-disciplines within computer science have developed a collection of abstractions that both (a) represent how we think about the important ideas of that sub-discipline and (b) provide guidance for how software that implement those ideas may be written. To a great extent the goal of this conference is to explore the more general abstractions that allow abstractions themselves to be represented in and manipulated by executable software.

Acknowledgment. I owe much to discussions with and the support of Debora Shuger.

References


1. Abbott, R.: Emergence Explained: Abstractions: Getting epiphenomena to do real work. *Complexity* 12(1), 13–26 (2006)
2. Abbott, R.: If a tree casts a shadow is it telling the time. In: Calude, C.S., Dinneen, M.J., Păun, G., Rozenberg, G., Stepney, S. (eds.) UC 2006. LNCS, vol. 4135, pp. 41–56. Springer, Heidelberg (2006) (A revised version to appear in the *International Journal of Unconventional Computation*.)

What's Your Problem?

The Problem of Problem Definition

J. Christopher Beck and Michael Gruninger

Department of Mechanical & Industrial Engineering
University of Toronto
{jcb,gruninger}@mie.utoronto.ca

Abstract. The first step in developing an application to solve a real world problem is to define the problem. Typically in applied mathematics, artificial intelligence, and operations research, the definition process generates a well-defined problem that is subsequently studied and, if the project is successful, solved (for some appropriate definition of “solved”). Our thesis is that problem definition is inherently a process of abstraction, reformulation, and approximation that has not been deeply studied in the literature. 

1 A Definition of Problem Definition

Consultants and applied researchers are often faced with real-world objects: a chocolate-bar factory or a set of operating theatres in a hospital to schedule, a classroom timetable at a university to develop. For all but the most trivial such problems, the consultant is faced with deciding what aspects of the situation should be included in a problem definition. This is a critical set of decisions because the problem definition has a great influence on the computational complexity of the eventual mathematical model of the problem as well as the extent to which a solution to the problem is actually useful in the real world.

The problem definition process is inherently one of abstraction, approximation, and, because it is likely to be iterative, reformulation. Can a production facility ignore the variance in activity durations? Can it account for machine breakdown by only scheduling machines at 80% capacity? Can we ignore tooling, the scheduling of human operators, upstream and downstream facilities? These issues are difficult to resolve because the impact of the decisions are inter-dependent and may become known much later in the development process (or never). Typically, it comes down to experience and, perhaps, small-scale experiments with prototype systems.

Problem Definition \neq Mathematical Modeling. The problem of problem definition as sketched above is not the same as the “modeling problem” that has been extensively studied in constraint programming and operations research. Modeling begins with a complete problem definition and develops and compares different mathematical models to solve the problem so defined.

¹ This paper is less a research summary and more a description of what we think is an interesting research direction. References to relevant work are solicited.

2 Three Orthogonal Research Directions

An Experimental System. We do not know very much about the impact of problem definition decisions on application success or failure. One, low-level direction, is to develop a detailed simulation model of the real object such as a factory and use this as the “real world” for experimentation². Given the simulation model, we could then develop different problem definitions at different levels of abstraction and, crucially, evaluate the impact of the levels of abstraction by solving the optimization models and “executing” the solutions in the simulation model. The primary question to investigate is the trade-off between the quality of the executed solution vs. the computational complexity of finding it and how the problem definition affects this trade-off.

Ontologies, Domain Modeling, and Problem Definition. An ontology is a set of reusable, sharable logical definitions that can be used for knowledge representation. With an ontology, such as Process Specification Language (PSL) [11], it is possible to build a logical domain model of a given real-world problem. Nevertheless, the specification of a domain model is not easy; in particular, the problem definition must be addressed: at what level of abstraction should the various real-world entities be represented? The answer, of course, is: at whatever level is sufficient for the application. Sufficiency, however, must be judged by the outcome of the overall system. And so we are left, again, to rely on judgment and experience.

However, an empirical approach may be possible. Imagine a set of applications in a given area, such as production scheduling, that have both software systems and accompanying domain models. Based on differences (and changes during development) in the software systems, the corresponding models, and the corresponding outcomes, we may be able to begin to develop a predictive or advisory meta-system. Given a domain model of a new production scheduling problem, the meta-system, by reasoning about existing models and outcomes, may be able to provide predictions about performance outcomes, advice on abstractions or refinements that might be useful, recommendations for optimization technology, and “what-if” analysis.

A Methodology for System Engineering. Finally, given that the problem definition problem appears AI-complete, it is likely to be a human activity for the foreseeable future. The abstraction, approximation, and reformulation that is inherent in defining and building software systems in general would seem to apply equally to optimization systems³. Building on software engineering, perhaps a methodology for system engineering can be developed, incorporating ontologies and simulation prototypes.

References

1. Gruninger, M.: Ontology of the process specification language. In: Staab, S., Studer, R. (ed.) *Handbook of Ontologies*, pp. 575–592 (2003)

² The simulation model also requires a problem definition process. The hope is that it will be more detailed and refined than any optimization model we would investigate.

³ Is building an optimization system different than building any other software system?

A Reformulation-Based Approach to Explanation in Constraint Satisfaction

Hadrien Cambazard and Barry O’Sullivan

Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
{h.cambazard,b.osullivan}@4c.ucc.ie

1 Introduction

Many approaches to explanation generation have been reported in the literature [1–5]. The dominant approach to explanation is based on computing minimal conflicting sets of constraints. An explanation can be considered concise if it involves few constraints of low arity. However, in many practical domains constraints are specified extensionally as *table constraints*. From an explanation point of view, table constraints can prevent us from finding concise explanations since the constraint may be specified over all the variables in the problem. However, it is often possible to reformulate a large arity table constraint into a set of low arity constraints whose join is logically equivalent to the original table constraint. In this research we are concerned with finding reformulations of large arity table constraints into low arity constraints in order to help find more concise explanations in interactive applications such as product configuration or online electronic commerce.

Example 1 (Reformulating a Positively Defined Table Constraint). Consider the following table constraint which is defined by a set of allowed tuples:

$$C(x_1, x_2, x_3, x_4) \equiv \{(0, 0, 0, 4), (1, 0, 0, 2), (2, 4, 1, 3), (0, 4, 2, 4), (2, 2, 3, 2)\}.$$

The conjunction of the following constraints is logically equivalent to C :

$$\begin{aligned} C_1(x_1, x_3) &\equiv \{(0, 0), (1, 0), (2, 1), (0, 2), (2, 3)\}, \\ C_2(x_1, x_4) &\equiv \{(0, 4), (1, 2), (2, 3), (2, 2)\}, \\ C_3(x_2, x_3) &\equiv \{(0, 0), (4, 1), (4, 2), (2, 3)\}, \\ C_4(x_2, x_4) &\equiv \{(0, 4), (0, 2), (4, 3), (4, 4), (2, 2)\}. \end{aligned}$$

In the next section we explain how we compute the decomposition in this example. ▲

2 Reformulation for Explanation

We identify suitable reformulations of table constraints by exploiting functional dependencies in the relation (the set of tuples) of the constraint. Functional dependencies express the presence of structure in relations that can be useful to improve the design of a database [6]. Given a relation R , a set of attributes X in R is said to functionally

determine another attribute y , also in R , written $X \rightarrow y$, if and only if each X value is associated with at most one y value. One can verify that $\{x_1, x_2\} \rightarrow x_4$ and $\{x_3\} \rightarrow x_2$ are two functional dependencies that hold in constraint C of Example 1. Using the first dependency, one can decompose C into: $C_a(x_1, x_2, x_4)$ and $C_b(x_1, x_2, x_3)$. Notice, however, that x_1 and x_2 determine x_4 independently, so the following decomposition is also valid: $C_2(x_1, x_4), C_4(x_2, x_4)$ and $C_b(x_1, x_2, x_3)$. Then by applying $\{x_3\} \rightarrow x_2$ to C_b we get the final binary decomposition described in Example 1. Functional dependencies can be determined using tools such as `tane`¹. Approximate dependencies can also be found, and are also useful for reformulating table constraints. We give an example of the power of our approach for generating concise explanations.

Example 2 (Computing Concise Explanations using Functional Dependencies). Consider the following three (positively defined) table constraints:

$$\begin{aligned} C_1(x_1, x_2, x_4, x_5) &\equiv \{(0, 1, 1, 1), (1, 2, 2, 0), (1, 0, 0, 2), (2, 1, 1, 1)\}, \\ C_2(x_1, x_3, x_6, x_7) &\equiv \{(2, 2, 0, 1), (1, 1, 0, 2), (1, 1, 2, 0), (2, 2, 1, 1)\}, \\ C_3(x_2, x_3, x_5, x_7) &\equiv \{(0, 2, 1, 2), (1, 0, 2, 1), (1, 1, 2, 2), (2, 0, 0, 1)\}. \end{aligned}$$

This set of constraints is inconsistent. However, an explanation in the form of a minimal conflict will contain all three 4-ary constraints. Constraint C_1 can be decomposed, for example, from the following functional dependencies: $\{x_2\} \rightarrow x_4$ and $\{x_2\} \rightarrow x_5$. Using a reformulation based on functional dependencies we can decompose each of the constraints as follows:

$$\begin{aligned} C_1(x_1, x_2, x_4, x_5) &\equiv C_{11}(x_1, x_2) \wedge C_{12}(x_2, x_4) \wedge C_{13}(x_2, x_5), \\ C_2(x_1, x_3, x_6, x_7) &\equiv C_{21}(x_1, x_3) \wedge C_{22}(x_1, x_6, x_7), \\ C_3(x_2, x_3, x_5, x_7) &\equiv C_{31}(x_2, x_5) \wedge C_{32}(x_2, x_3) \wedge C_{33}(x_3, x_7). \end{aligned}$$

where the relations of the constraints on the right are obtained by projecting their scopes onto the relations of the constraints on the left. In this decomposition, we still have inconsistency and an explanation of that is $C_{11}(x_1, x_2) \wedge C_{21}(x_1, x_3) \wedge C_{32}(x_2, x_3)$, which is much more concise than the original explanation involving three 4-ary constraints. ▲

Acknowledgements. Supported by Science Foundation Ireland (Grant No. 05/IN/I886).

References

1. Amilhastre, J., Fargier, H., Marquis, P.: Consistency restoration and explanations in dynamic CSPs application to configuration. *Artif. Intell.* 135(1-2), 199–234 (2002)
2. Junker, U.: QuickXplain: preferred explanations and relaxations for over-constrained problems. In: *Proceedings of AAAI*, pp. 167–172 (2004)
3. O'Callaghan, B., O'Sullivan, B., Freuder, E.C.: Generating corrective explanations for interactive constraint satisfaction. In: *Proceedings of CP*, pp. 445–459 (2005)
4. O'Sullivan, B., Papadopolous, A., Faltings, B., Pu, P.: Representative explanations for over-constrained problems. In: *Proceedings of AAAI* (forthcoming) (2007)
5. Sqalli, M.H., Freuder, E.C.: Inference-based constraint satisfaction supports explanation. In: *Proceedings of AAAI*, pp. 318–325 (1996)
6. Ullman, J.D., Widom, J.D.: *First Course in Database Systems*. Prentice-Hall, Englewood Cliffs (2002)

¹ <http://www.cs.helsinki.fi/research/fdk/datamining/tane/>

Integration of Constraint Programming and Metaheuristics

Broderick Crawford^{1,2}, Carlos Castro², and Eric Monfroy^{2,3,*}

¹ Pontificia Universidad Católica de Valparaíso, Chile
FirstName.Name@ucv.cl

² Universidad Técnica Federico Santa María, Chile
FirstName.Name@inf.utfsm.cl

³ LINA, Université de Nantes, France
FirstName.Name@univ-nantes.fr

Abstract. Our research is focused on developing hybrid solvers for Combinatorial Optimization Problems. We are concerned with the design of hybrid resolution approaches including Constraint Programming and Metaheuristics. We have been working on that area during the last years, exploring the different issues involved in algorithm design, implementation, tuning and experimental evaluation. We provide an overview of the research we have completed as well as of the future work.

In order to be able to solve any combinatorial optimization problem it seems that a good idea is to use both incomplete and complete techniques together. When problems are easy enough to allow searching for the optimal solution, complete techniques can be used. When problems become harder, incomplete techniques represent a good alternative in order to solve approximately the problem. Particularly, promising possibilities of combining constraint programming (CP) and metaheuristics are pointed out in [\[9,6,11,5\]](#).

Indeed, a complete search can guide constructive metaheuristics: constraint propagation can be applied in order to restrict the neighborhood or prune the search space. Complete techniques are also used in order to explore the neighborhood of the current configuration selecting the next moves. Following these ideas, in [\[2,3\]](#), we solve some benchmarks of subset problems with ant colony optimization (ACO) algorithms and some hybridizations of ACO with CP. Here, a lookahead mechanism allows the incorporation of information on the anticipated decisions that are beyond the immediate choice horizon. The ants solutions may contain redundant components which can be eliminated by a fine tuning after the solution, then we explore post processing procedures too. Computational results are presented showing the advantages to use additional mechanisms to ACO. In [\[4\]](#), we apply the same approach focus on the resolution of crew pairing

* The second author has been partially supported by the Chilean National Science Fund through the project FONDECYT 1070268. The third author has been partially supported by the Chilean National Science Fund through the project FONDECYT 1060373.

optimization. We fix the difficulty of pure ant algorithms solving strongly constrained problems and we explore the addition of CP in the construction phase of the ants, so they can complete their solutions.

From other point of view, metaheuristics and local search algorithms can help a complete technique: in [8], we propose to use local search for guiding enumeration. We extend the common variable selection strategies of CP and we achieve the value selection based on a local search. Note that in constraint programming, enumeration strategies (selection of a variable and a value of its domain) are crucial for resolution performances.

At last, we plan to develop a software environment for prototyping hybrid algorithms using dynamic strategies based on CP and metaheuristics [7].

References

1. Castro, C., Moossen, M., Riff, M.C.: A Cooperative Framework Based on Local Search and Constraint Programming for Solving Discrete Global Optimisation. In: Bazzan, A.L.C., Labidi, S. (eds.) SBIA 2004. LNCS (LNAI), vol. 3171, pp. 93–102. Springer, Heidelberg (2004)
2. Crawford, B., Castro, C.: Improving the performance of Ant Algorithms using Constraint Programming. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 31–36. Springer, Heidelberg (2006)
3. Crawford, B., Castro, C.: Integrating Lookahead and Post Processing Procedures with ACO for Solving Set Partitioning and Covering Problems. In: Rutkowski, L., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2006. LNCS (LNAI), vol. 4029, pp. 1082–1090. Springer, Heidelberg (2006)
4. Crawford, B., Castro, C., Monfroy, E.: A constructive Hybrid Algorithm for Crew Pairing Optimization. In: Euzenat, J., Domingue, J. (eds.) AIMSA 2006. LNCS (LNAI), vol. 4183, pp. 45–55. Springer, Heidelberg (2006)
5. Focacci, F., Laburthe, F., Lodi, A.: Local Search and Constraint Programming. In: Handbook of metaheuristics, Kluwer Academic Publishers, Dordrecht (2002)
6. Meyer, B., Ernst, A.: Integrating ACO and Constraint Propagation. In: Dorigo, M., Birattari, M., Blum, C., Gambardella, L.M., Mondada, F., Stützle, T. (eds.) ANTS 2004. LNCS, vol. 3172, pp. 166–177. Springer, Heidelberg (2004)
7. Monfroy, E., Castro, C., Crawford, B.: Adaptive enumeration strategies and metabacktracks for constraint solving. In: Yakhno, T., Neuhold, E.J. (eds.) ADVIS 2006. LNCS, vol. 4243, pp. 354–363. Springer, Heidelberg (2006)
8. Monfroy, E., Castro, C., Crawford, B.: Using local search for guiding enumeration in constraint solving. In: Euzenat, J., Domingue, J. (eds.) AIMSA 2006. LNCS (LNAI), vol. 4183, pp. 56–65. Springer, Heidelberg (2006)
9. Monfroy, E., Saubion, F., Lambert, T.: Hybrid CSP Solving. In: Gramlich, B. (ed.) Frontiers of Combining Systems. LNCS (LNAI), vol. 3717, pp. 138–167. Springer, Heidelberg (2005) (Invited paper)

Rule-Based Reasoning Via Abstraction

Research Summary

David C. Haley
Stanford Logic Group

Stanford University, Computer Science Department
Gates 226, 353 Serra Mall, Stanford, CA 94305
dhaley@cs.stanford.edu

The *General Game Playing* (GGP) project seeks to create software agents capable of receiving game rules hitherto unseen and, without human interaction, play that game effectively. For the time being, we have chosen to restrict the class of games to be multi-player, finite, deterministic and complete information. (This is precisely the class of games that can be modeled by a finite-state machine.) Due to the finiteness and deterministic restrictions, given the rules of a game it is in principle possible to completely expand the game model and reason about it using model-checking or traditional graph search techniques. In practice, however, the games can be intractably large, making these techniques inadequate; games might also have special structure or properties implied by the rules, which if discovered could mean considerable shortcuts in the search. For this reason, we prefer reasoning about the game rules and only expand the game tree when absolutely necessary, using any insight gained from the rules.

In this research, we seek to use theorem proving and mathematical induction to verify claims about the game. This is building on previous work in the Stanford Logic Group by Timothy Hinrichs on proving playability¹ of a game by constructing an abstract representation of the state and proving that if some hypothesis holds, every player has a legal move, and then proving by induction that said hypothesis always holds. The abstract representation is constructed such that if playability holds in the abstract, then it holds in the original game. The advantage of using abstraction is that it reduces the size of the proof space, for example when performing an inductive proof.

Now, we would like to prove more than playability: if the truth of some hypothesis could lead to shortcuts in the search, we would like to prove that hypothesis. For example, we could show that the truth of a set of facts entails an eventual win or loss of a player, or that a set of facts entails that it is still possible but not guaranteed for a player to win. With this additional knowledge, the search could be pruned very aggressively by avoiding all states in which the ‘bad’ facts appear and favoring states in which the ‘good’ facts appear.

The research, then, consists in making hypotheses about the game that would help guide or prune the search, and checking them by building abstract representations of the game in which the proof search will be much less computationally expensive.

¹ The property that at every reachable state, every player has at least one move.

Extensional Reasoning

Timothy L. Hinrichs

Stanford University
thinrich@cs.stanford.edu

Abstract. Relational databases are one of the most industrially successful applications of formal logic in computer science. The power of the paradigm is clear both because of its widespread adoption and because of theoretical analysis. Today, automated theorem provers are not able to take advantage of database systems and therefore do not routinely leverage that source of power. Extensional Reasoning is an approach to automated theorem proving where the machine automatically translates a logical entailment query into a database, a set of view definitions, and a database query so that the entailment query can be answered by answering the database query. In some cases this approach produces several orders of magnitude performance improvement over traditional theorem proving techniques.

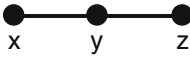
Relational databases are one of the most successful applications in computer science, yet today's theorem provers fail to capitalize on that success. Extensional Reasoning (ER) is an approach to automated theorem proving where the system transforms a logical entailment query into a database (a set of extensionally defined tables), a set of view definitions (a set of intensionally defined tables), and a database query.

ER was developed because many problems can be solved efficiently using a database but are most naturally expressed using classical logic. By allowing users the ability to write problems down in a natural way, there are fewer encoding errors; moreover, a system that allows more natural encodings is accessible to users who are not necessarily experts in automated reasoning, e.g. machines.

For example, the map coloring problem is often stated as follows: given a map and a set of colors, paint each region of the map so that no two adjacent regions are the same color. One very natural way to encode this problem (according to the introductory logic students at Stanford who offer up this formulation year after year) centers around the following constraint.

$$color(X, C) \wedge adj(X, Y) \Rightarrow \neg color(Y, C)$$

When studied in the deductive database/logic programming community [1], a different formulation is used. Fig. 1 shows a map with three regions and the corresponding database formulation using the colors *red* and *blue*. In Extensional Reasoning, this transformation happens automatically, which allows all the results from (deductive) databases and logic programming to be brought to bear on the problem.



Query: $next(X, Y) \wedge next(Y, Z)$

Database:

<i>next</i>	
red	blue
blue	red

Fig. 1. A 3-node graph and its corresponding database query

The techniques for performing Extensional Reasoning can be partitioned into two classes: those for complete theories and those for incomplete theories. A complete theory corresponds very naturally to a relational database because there is essentially a single model to consider for the purpose of entailment. Transforming a complete theory into a database amounts to choosing the portion of that model to represent extensionally as the database, leaving the remainder to be represented intensionally, as view definitions. Complete theories are rare, and recognizing that a theory is complete can be nontrivial, but sometimes the work is worth the effort, as the improvement in performance can be several orders of magnitude.

Complete theories have powerful properties, but incomplete theories are the norm. Transforming an incomplete theory into a database is problematic because there are several models that must be taken into account for the purpose of entailment. One could transform each model into a database system and check that whether the database query is true in every database system, implementing the definition of entailment directly, but because the number of models grows exponentially, this approach is prohibitively expensive. The reformulation technique illustrated by the map coloring example collapses models together, with the hope that only the important differences between models are enumerated. This type of reformulation lies at the heart of Extensional Reasoning for incomplete theories.

While we hope Extensional Reasoning will eventually be applied to a wide variety of logics, for the time being we have elected to focus on theories in a decidable logic, placing the issue of efficiency front and center. The particular logic studied thus far is a fragment of first-order logic that is a perennial problem in the theorem proving community: it includes the domain closure axiom, which guarantees decidability while allowing arbitrary quantification. This logic, to which the map coloring example belongs, allows us to avoid issues of undecidability at this early stage in the development of Extensional Reasoning, while at the same time giving us the opportunity to make progress on an important class of problems.

References

1. McCarthy, J.: Coloring maps and the Kowalski doctrine. Stanford Technical Report (1982)

Reformulating Constraint Models Using Input Data^{*}

Martin Michalowski¹, Craig A. Knoblock¹, and Berthe Y. Choueiry^{1,2}

¹ University of Southern California, Information Sciences Institute
4676 Admiralty Way, Marina del Rey, CA 90292 USA
{martinm,knoblock}@isi.edu

² Constraint Systems Laboratory, University of Nebraska-Lincoln
choueiry@cse.unl.edu

1 Motivation

Consider the problem of mapping postal addresses to buildings in satellite imagery using publicly available information, defined as the Building Identification (BID) problem in [1]. This problem takes as input a bounding box that defines the area of a satellite image, buildings identified in the image, vector information that specifies streets in the image, and a set of phone-book entries for the area. The task is to find the set of possible address assignments for each building. In [1], we showed how the task can be framed as a Constraint Satisfaction Problem (CSP), which we solved with an existing solver in [1] and a custom solver in [2]. The CSP is given by $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ where \mathcal{V} is the set of buildings, \mathcal{D} the set of their respective potential addresses, and \mathcal{C} a set of constraints that describe the physical layout of the buildings on the map and address numbering strategies.

In the context of a web application, a typical BID scenario is as follows. A user, presented with a map such as a Google map, either selects a specific building in an area of interest and requests the address of the building, or he/she provides an address and requests the buildings that could have this address. This process is repeated for millions of areas throughout the United States. To answer the entire spectrum of user queries, this application needs to contend with the slight addressing variations found in cities across the US. For example, some cities adhere to a block numbering scheme where addresses increment by a fixed factor (i.e., 100 or 1000) across street blocks while others do not. The direction in which addresses increase also varies, in some cities this occurs to the east while in others it is to the west. In other cities, addresses along East-West running streets increase to the West in one part of town but to the East in the other. Finally, expanding this application to support the rest of the world would require the set of constraints to model new addressing characteristics not seen in the US. The globalization of addressing across continents ensures that some general guidelines are followed, but this standardization is typically met with regional/cultural customization such as the districting in Venice or the historical numbering seen in Japan. The creation of individual models, for each city in the

* This research is supported in part by the Air Force Office of Scientific Research under grant numbers FA9550-04-1-0105 and FA9550-07-1-0416.

world, that account for all of the addressing constraints is an overwhelming and unrealistic task. However, the work required of the expert to define constraints that capture all of the characteristics of addressing seen to date is relatively small and manageable. We propose a framework in which the constraint model of the area of interest for a given user is dynamically built by augmenting the set of *basic* constraints, which form the *generic* constraint model, with those constraints that specify the addressing schema that governs the area of interest.

2 Research Approach

We propose to exploit the information found within a problem instance to enrich the generic model of the CSP in order to identify the set of constraints that apply in a given setting, as shown in Fig. 1.

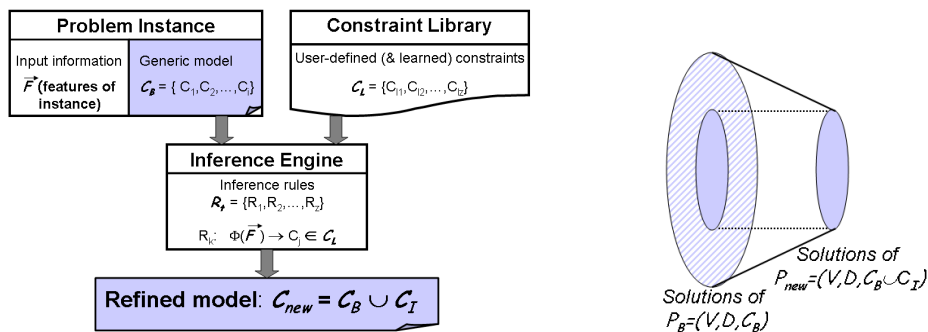


Fig. 1. *Left:* Building the customized constraint model from the generic one. *Right:* Comparing the solution sets of the generic & customized models.

The embedded information that we exploit is a set of instantiated variables, which we call *data points*. Our framework tests the features \vec{F} of these data points in order to select, from a library of constraints \mathcal{C}_L , those constraints \mathcal{C}_I that should be added to the generic constraint model \mathcal{C}_B of the problem. We reduce the load on a human user by limiting their involvement to defining the library of constraints, which is leveraged over the repetitive use of the application over various areas. Subsequently, we use the expert knowledge introduced by the user along with the information found in the problem description to generate a customized problem model that best represents the problem instance at hand. This approach enables a more flexible approach to dynamically modeling problem instances by reformulating problem models and not requiring a collection of individual models that represent all of the foreseeable variations of a problem class. The set of constraints $\mathcal{C}_{new} = \mathcal{C}_B \cup \mathcal{C}_I$ allows us to approach the most accurate model and return more precise solutions (see Fig. 1). We also use constraint propagation on \mathcal{C}_B in order to infer new data points.

References

1. Michalowski, M., Knoblock, C.A.: A constraint satisfaction approach to geospatial reasoning. In: Proc. of AAAI-05, pp. 423–429 (2005)
2. Bayer, K., Michalowski, M., Choueiry, B.Y., Knoblock, C.A.: Reformulating constraint satisfaction problems to improve scalability. In: Proc. of SARA-07 (2007)

Using Analogy Discovery to Create Abstractions

Marc Pickett

Cognition, Robotics, and Learning
University of Maryland, Baltimore County
marc@coral.cs.umbc.edu

Concept formation is a form of abstraction that allows for knowledge transfer, generalization, and compact representation. Concepts are useful for the creation of a generally intelligent autonomous agent. If an autonomous agent is experiencing a changing world, then nearly every experience it has will be unique in that it will have at least slight differences from other experiences. Concepts allow an agent to generalize these experiences and other data. In some applications, the concepts that an agent uses are explicitly provided by a human programmer. A problem with this approach is that the agent encounters difficulties when it faces situations that the programmer had not anticipated. For this reason, it would be useful for the agent to automatically form concepts in an unsupervised setting. The agent should be able to depend as little as possible on representations tailored by humans, and therefore it should develop its own representations from raw uninterpreted data.

One purpose of concept formation (and abstraction in general) is to concisely characterize a set of data [7]. With this view, one can use *minimum description length* as a guiding principle for concept formation. My research uses this principle to form an ontology of concepts from a collection of data. This data is a set (or a stream) of statements, where each statement is an ordered tuple of symbols (representing relations). The symbols have no meaning for the program other than they're considered to be ground statements. For example, these symbols can be raw sensor data, or raw descriptions of chess games.

For SARA 2005, Tim Oates and I developed an ontology formation algorithm called The Cruncher [6] that works on attribute-value data. The Cruncher (an extension of PolicyBlocks [5], an algorithm for discovering useful macro-actions in Reinforcement Learning that I developed with Andy Barto) is a simple representation framework and algorithm based on minimum description length for automatically forming an ontology of concepts from attribute-value data sets. Although unsupervised, when The Cruncher is applied to the Zoo database from [1], it produces a nearly zoologically accurate categorization. The Cruncher can also be applied to find useful macro-actions in Reinforcement Learning, learn models from uninterpreted sensor data, or form an ontology of documents based on word-frequency.

It's useful to be able to develop *relational* concepts through *analogy*. Some suggest that analogy may even be the "core of cognition" [3]. Analogy allows us to focus on the relations among entities rather than superficial aspects of the entities. For example, we might notice that a red ant killing a black ant and stealing a piece of food it is analogous to a situation in Hamlet where Claudius murders Hamlet's father and usurps the throne of Denmark. In this situation, we

must also be able to specify that the red ant corresponds to Claudius, the black ant to Hamlet's father, and the piece of food maps to the throne. Once found, relational concepts can be useful for knowledge transfer: conclusions about one domain can map to another domain.

Currently, I'm extending The Cruncher to work on relational data. The extended algorithm, The Übercruncher, discovers isomorphisms (or analogies) in relational data, and forms concepts from the analogies to compress the data. After finding a set of analogies, the best analogy is used to compress the Knowledge Base, resulting in a shorter description. This entire process (finding analogies and crunching with them) is repeated until no more useful analogies are found. In practice, useful analogies are often found as parts of other analogies, which produces a multi-tiered ontology.

The Übercruncher is related to the SUBDUE system [4] which compresses graphs by finding common substructures. Both SUBDUE and The Übercruncher work on data that's not presegmented, and both use minimum description length as the guiding principle by which substructures are evaluated. Like The Übercruncher, SUBDUE also does *induction* in the sense that frequently occurring substructures are replaced by a node that symbolizes the full substructure. However, SUBDUE uses a potentially slow beam search, upon which The Übercruncher improves by building a conceptual structure that can be used to accelerate learning and classification into a current ontology. Additionally, The Übercruncher represents both concepts and meta-concepts in the same framework so that the same algorithm can be used to find analogies in both data and meta-data. Ignoring differences in representation and search strategy, SUBDUE is essentially a strictly bottom-up version of The Übercruncher, which also uses top-down guidance for classification, similar to that described by [2].

Future work involves developing a full cognitive architecture, dubbed The Marchitecture, that *uses* the ontology developed by The Übercruncher for reasoning, planning, classification, and explanation, and integrates these processes with formation of the ontology.

References

1. Blake, C.L., Merz, C.J.: UCI repository of machine learning databases (1998)
2. Hawkins, J., Blakeslee, S.: On Intelligence. Times Books (2004)
3. Hofstadter, D.R.: Analogy as the core of cognition. The Analogical Mind: Perspectives from Cognitive Science, pp. 499–538 (2001)
4. Holder, L., Cook, D., Djoko, S.: Substructure discovery in the subdue system. In: Proceedings of the Workshop on Knowledge Discovery in Databases (1994)
5. 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)
6. Pickett, M., Oates, T.: The cruncher: Automatic concept formation using minimum description length. In: Zucker, J.-D., Saitta, L. (eds.) SARA 2005. LNCS (LNAI), vol. 3607, Springer, Heidelberg (2005)
7. Wolff, J.G.: Information compression by multiple alignment, unification and search as a unifying principle in computing and cognition. Artif. Intell. Rev. (2003)

Distributed CSPs: Why It Is Assumed a Variable per Agent?*

Miguel A. Salido

DSIC, Technical University of Valencia
Camino de Vera s/n, 46071, Valencia, Spain
msalido@dsic.upv.es

Abstract. Nowadays, many real problems can be formalized as Distributed CSPs. A distributed constraint satisfaction problem (DisCSP) is a CSP in which variables and constraints are distributed among multiple automated agents. Many researchers assume for simplicity that each agent has exactly one variable. For real distributed problem these techniques require a large amount of messages passed among agents, so these problems are very difficult to solve. In this research summary, we question why the lack of works to manage large-scale problems.

Keywords: Distributed Constraint Satisfaction Problems.

1 Introduction

In recent years we have seen an increasing interest in Distributed Constraint Satisfaction Problem (DisCSP) formulations to model combinatorial problems (see the special issue on Distributed Constraint Satisfaction in Artificial Intelligence journal, vol 161, 2005). There is a rich set of real-world distributed applications, such as networked systems, planning, scheduling, etc, for which the DisCSP paradigm is particularly useful. In such distributed applications, privacy issues, knowledge transfer costs, robustness against failure, etc preclude the adoption of a centralized approach [3].

A distributed CSP is a CSP in which the variables and constraints are distributed among automated agents. Finding a value assignment to variables that satisfies inter-agent constraints can be viewed as achieving coherence or consistency among agents.

The more cited papers related to DisCSP make the following assumptions for simplicity in describing the algorithms:

1. Each agent has exactly one variable.
2. All constraints are binary.
3. Each agent knows all constraint predicates relevant to its variable.

* This work has been partially supported by the research projects TIN2004-06354-C02-01 (Min. de Educacion y Ciencia, Spain-FEDER) and GV/2007/274 (G. Valenciana).

Although the great majority of real problems are naturally modelled as non-binary CSPs, the second assumption is comprehensible due to there exist some techniques that translate any non-binary CSP into a equivalent binary one [1].

However, the first assumption is too restrictive and the main basic research is focused to small instances and little work has been done to solve real-life problems.

2 From Basic Research Toward Applied Research

One of the pioneer researchers in DisCSP said "So far, we assume that each agent has only one local variable. Although the developed algorithms can be applied to the situation where one agent has multiple local variables by the following methods, both methods are neither efficient nor scalable to large problems" [5].

- Method 1: each agent finds all solutions to its local problem first. By finding all solutions, the given problem can be re-formalized as a distributed CSP, in which each agent has one local variable whose domain is a set of obtained local solutions. Then, agents can apply algorithms for the case of a single local variable. The drawback of this method is that when a local problem becomes large and complex, finding all the solutions of a local problem becomes virtually impossible.
- Method 2: an agent creates multiple virtual agents, each of which corresponds to one local variable, and simulates the activities of these virtual agents. However, since communicating with other agents is usually more expensive than performing local computations, it is wasteful to simulate the activities of multiple virtual agents without distinguishing the communications between virtual agents within a single real agent, and the communications between real agents.

In spite of significant progress in distributed CSP, the following question is straightforward: *Why it is assumed a Variable per Agent?*

Only some works include a set of variables into an agent [4], [2]. Therefore, more research must be done to solve more realistic problems.

References

1. Bacchus, F., van Beek, P.: On the conversion between non-binary and binary constraint satisfaction problems. In: proceeding of AAAI-98, pp. 311–318 (1998)
2. Bessi ere, C., Bellaissaoui, M., Ezzahir, R., Bouyakhf, El-H.: Dischoco: A platform for distributed constraint programming. In: Proceedings of IJCAI-07 Workshop on Distributed Constraint Reasoning (2007)
3. Faltings, B., Yokoo, M.: Introduction: Special issue on distributed constraint satisfaction. Artificial Intelligence 161, 1–5 (2005)
4. Salido, M.A., Barber, F.: Distributed CSPs by graph partitioning. Applied Mathematics and Computation 183, 491–498 (2006)
5. Yokoo, M., Hirayama, K.: Algorithms for distributed constraint satisfaction: A review. Autonomous Agents and Multi-Agent Systems 3, 185–207 (2000)

Decomposition of Games for Efficient Reasoning

Eric Schkufza

Stanford University
eschkufz@stanford.edu

Abstract. General Game Playing (GGP) is an area of research in artificial intelligence that focuses on the representation of games in terms of an abstract language known as Game Description Language (GDL). The abstract nature of that language allows for the development of intelligent agents that without modification can perform competently on games that they have never seen before based on known properties of GDL's uniform and compact syntax. Although GDL is an effective tool for communication, it is less useful as a representational framework for reasoning about games. In its place, the Stanford Logic Group has developed a new class of behavioral models, called Propositional Nets (PNs), with which an algorithm has been designed for determining whether games can be decomposed into independent sub-systems that can be reasoned about independently of one another.

GGP focuses on discrete, multi-player, deterministic, complete-information games. GDL represents game states by keeping track of the set of propositions that are true of the world (part of the description of tic-tac-toe for instance, might be a proposition used to represent whether or not there is an X in the center cell). GDL represents game dynamics as sets of actions that players can perform. Players are each permitted to make one move per turn, where the legality of a move is a function of the propositions that are true prior to its being made (in tic-tac-toe, a player may place an X in the center square if he is playing the role of X, it is his turn, and the square is currently empty).

Perhaps the most natural representational formalism that might be used to reason about the type of games that can be expressed in GDL is a finite state machine (FSM). However, FSMs have certain properties that make them inappropriate for use by general game playing systems. First, the number of states required to describe an FSM is in general exponential in the number of logical propositions required to encode each of those states. Thus, for all but the simplest games encoded in GDL, the size of the corresponding FSM would be intractably large. Second, FSMs are highly uniform structures; to describe a system, each state must in general associate a truth value to each of the propositions required to exhaustively describe it. Thus, even if a game encoded in GDL were to exhibit substantial independence and decomposability of substructures, it is unclear how that structure might efficiently be uncovered from an exponentially large set of states and a transition function alone.

Propositional Nets were designed by the Stanford Logic Group to address the above shortcomings. A PN is a bipartite graph consisting of nodes used to

represent propositions alternating with either logical connectives (AND, OR, or NOT gates) or transitions. The propositions in a PN can be partitioned into three classes: input propositions, whose truth values can be set by agents, base propositions, whose truth values are a function of incoming connections from transitions, and view propositions, whose truth values are a function of incoming connections from logic gates. The state of a PN is the set of truth values associated with its base propositions. Similarly, the dynamics of a PN are defined by the transitions that serve as inputs to those base propositions. State updates are accomplished by setting base propositions to be true if and only if the transitions that they are connected to have an incoming connection from a proposition that is true just prior to that update. Unlike FSMs, PNs are representationally compact; the number of propositions required to encode a system is linear in the number of facts required to represent the state of that system. Furthermore, the graphical structure of PNs makes it a straightforward process to determine the functional relationship between propositions; the consequences of changing the truth value of a proposition can be determined by simply following the outgoing arcs that emanate from it.

If a game expressed in GDL can be translated into a PN, then the topological structure of that PN can be used to perform a computationally efficient analysis of whether or not some of the propositions that it contains are independent of one another. Specifically, if there is no directed path between two propositions, then they can be reasoned about independently of one another, as neither one's truth value can ever affect the other's. Furthermore, if it can be shown that that PN is composed of two or more entirely disjoint subgraphs, then each of those subgraphs can be reasoned about independently of each other, as separate games. The potential computational savings associated with such a discovery are substantial. Consider the case of a game expressed in GDL that involves the simultaneous play of two games, one with m legal moves, and the other with n , both of which last for t moves. The complete search tree for such a game would contain $(mn)^t$ fringe nodes. However, if such a discovery were made and those games were considered separately from one another, then that search space could be reduced to one of two trees containing a combined $m^t + n^t$ fringe nodes.

The results described above suggest that PNs are a representational framework deserving of continued attention. Future research in the area will include further investigation of the mathematical properties of PNs as well as continued development of algorithms along the lines of the one presented in this summary.

References

1. Love, H., Genesereth.: General Game Playing: Game Description Language Specification. Technical report, Stanford University (2006)

Generalized Constraint Acquisition

Xuan-Ha Vu and Barry O’Sullivan

Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
{ha.vu,b.osullivan}@4c.ucc.ie

1 Introduction

Constraint programming is an approach to problem solving that relies on a combination of inference and search to solve real-world problems formulated as *constraint satisfaction problems* (CSPs). Many methods for solving CSPs have been developed. However, the specification of a CSP is sometimes not available, but may have to be learned from a *training set*, which is given, for instance, as a set of examples of its solutions and non-solutions. The motivating applications for constraint acquisition are many. For example, often one may wish to find a compact representation of a CSP instance for purposes such as explanation generation, requirements gathering, and specification. Acquiring soft constraints, which we focus on here, can be regarded as learning about preferences, uncertainty or costs in a combinatorial setting.

2 Related Work

One may apply techniques from the field of *machine learning* to acquire an appropriate formulation of a problem as a CSP. However, generic machine learning methods do not take the characteristics of CSPs into account, and are therefore often inefficient for acquiring CSPs. As a result, a new class of learning methods with emphasis on the characteristics of CSPs, called *constraint acquisition*, has been studied in recent years to learn CSPs. In [1], a specialized instance of the Candidate-Elimination learning method [2] has been devised to learn classical CSPs, where the *hypothesis space* of CSPs is maintained by exploiting a partial order over constraints. This algorithm has been improved further by exploiting constraint redundancy in CSPs [3]. Later, in [4], the CSP acquisition task has been reformulated as a satisfiability problem and solved more efficiently using SAT techniques.

3 Our Approach

Existing constraint acquisition methods, though useful for finding CSPs that fully agree with the training set, are limited to learning classical CSPs only and provide no useful information in the case where a CSP that fully agrees with the training set does not exist in the hypothesis space. This deficiency has motivated us to develop a new framework in which one can acquire CSPs

that minimally differ, according to a predefined measure, from the training set. Moreover, different types of constraints have been generalized into a unifying concept, the *semiring-based framework for soft constraints*, which allows one to extend many existing techniques for solving different types of CSPs in a standard and uniform way [5,6]. This has inspired us to develop a new unifying framework for constraint acquisition based on the concept of semiring-based soft constraints. In combination with the framework of semiring-based constraint satisfaction, our new framework enables users to exploit constraint satisfaction, optimization and acquisition techniques in a uniform way, thus reducing the effort required to develop analogous techniques for different classes of constraints.

The main objective of our research is to provide a general framework to unify acquisition algorithms for different types of constraints. In the past, acquisition algorithms were developed in an ad-hoc manner. Using our proposed framework, we can get the same, or even better, formulations by simply instantiating the framework.

Our new framework allows us to (i) uniformly formulate constraint acquisition problems as optimization problems; (ii) concisely derive existing constraint acquisition techniques from the framework; and (iii) formulate new, more general, constraint acquisition techniques as optimization problems. The framework is generic: it can be instantiated to obtain specific formulations for acquiring classical, fuzzy, weighted, or probabilistic CSPs. Specially, a new formulation for classical constraint acquisition with tolerance for violation, which allows us to find all CSPs that minimize the number of examples violated by them, can be developed as an instance of the framework. This formulation is equivalent to a simple *pseudo-Boolean optimization* (PBO) problem, which can be efficiently solved using many available tools.

Acknowledgements. This work was supported by the Embark Initiative and Science Foundation Ireland (Grant Number 05/IN/I886).

References

1. Coletta, R., Bessiere, C., O’Sullivan, B., Freuder, E.C., O’Connell, S., Quinqueton, J.: Constraint Acquisition as Semi-Automatic Modeling. In: Gedeon, T.D., Fung, L.C.C. (eds.) AI 2003. LNCS (LNAI), vol. 2903, pp. 111–124. Springer, Heidelberg (2003)
2. Mitchell, T.M.: Generalization as Search. *Artificial Intell.* 18(2), 203–226 (1982)
3. Bessiere, C., Coletta, R., Freuder, E.C., O’Sullivan, B.: Leveraging the Learning Power of Examples in Automated Constraint Acquisition. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 123–137. Springer, Heidelberg (2004)
4. Bessiere, C., Coletta, R., Koriche, F., O’Sullivan, B.: A SAT-Based Version Space Algorithm for Acquiring Constraint Satisfaction Problems. In: Fürnkranz, J., Schefler, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 23–34. Springer, Heidelberg (2006)
5. Bistarelli, S., Montanari, U., Rossi, F.: Constraint solving over semirings. In: *IJCAI* (1), pp. 624–630 (1995)
6. Bistarelli, S., Montanari, U., Rossi, F.: Semiring-Based Constraint Satisfaction and Optimization. *Journal of the ACM* 44(2), 201–236 (1997)

Using Infeasibility to Improve Abstraction-Based Heuristics

Fan Yang, Joseph Culberson, and Robert Holte

Computing Science Department, University of Alberta
Edmonton, Alberta T6G 2E8 Canada
{fyang, joe, holte}@cs.ualberta.ca

The contribution of our research is to show that the accuracy of the heuristics generated by abstraction can be improved by checking for infeasibility. What do we mean by infeasible heuristics? For a state t , the heuristic value h is infeasible if it is proved that the cost of a solution for t cannot be h . Take the sliding puzzle for example, assuming that the manhattan heuristic for state t is $md(t)$, if $md(t)$ is even, any odd number is infeasible. To substantiate our approach, we begin with formal definitions and lemmas. Then empirical results show the effectiveness of the approach. For more details please refer to our longer work [5].

A *state space* is a weighted directed graph with a set of states, a set of directed edges (ordered pairs of states) and the edge cost function. For example, a set of states may be defined by the set of all possible assignments to a set of state variables, and the edges and the edge cost function will depend on the operations on the variable sets. An *abstraction system* includes a state space, a set of abstract state spaces and a set of mappings $\Psi = \{\psi_1, \dots, \psi_k\}$ from the initial state space to abstract spaces. Our definition is similar to the work of Prieditis [4]. The key difference is that here we split each edge cost into two costs: the *primary cost* C_i and a *residual cost* R_i . Given a path p from t to g in the initial state space, for each abstract space \mathcal{A}_i , \mathbf{p}_i is the corresponding abstract path from t_i to g_i , where $t_i = \psi_i(t)$ and $g_i = \psi_i(g)$. To guarantee admissibility, we require that for any path p from t to g , $C(p) \geq C_i(\mathbf{p}_i) + R_i(\mathbf{p}_i)$. We say that abstractions are additive if the cost of each edge in the original space is larger than or equal to the sum of C_i of corresponding edges in all abstract state spaces. This definition generalizes those in [1, 2, 3, 4]. $C_i^*(t_i, g_i)$ is the minimum primary cost of an abstract path from t_i to g_i . Define $R_i^*(t_i, g_i)$ to be the minimum residual cost among the paths whose primary cost is minimal. Given a goal state g , the heuristic of state t defined by k additive abstractions is $h(t) = \sum_{i=1}^k C_i^*(t_i, g_i)$. Lemma 1 gives a test for infeasibility of additive abstraction-based heuristics.

Lemma 1. *Given k additive abstractions, if for some j , $1 \leq j \leq k$, we have $h(t) < C_j^*(t_j, g_j) + R_j^*(t_j, g_j)$, then $h(t)$ is infeasible.*

Figure 1 is an example test for infeasibility. Table 1 indicates that additive heuristics may be improved by checking for infeasibility. The performance of IDA* using additive heuristics with/without checking for infeasibility can be compared in the first two rows and the last two rows. The average running time of IDA* using the heuristics enhanced by checking for infeasible additive values

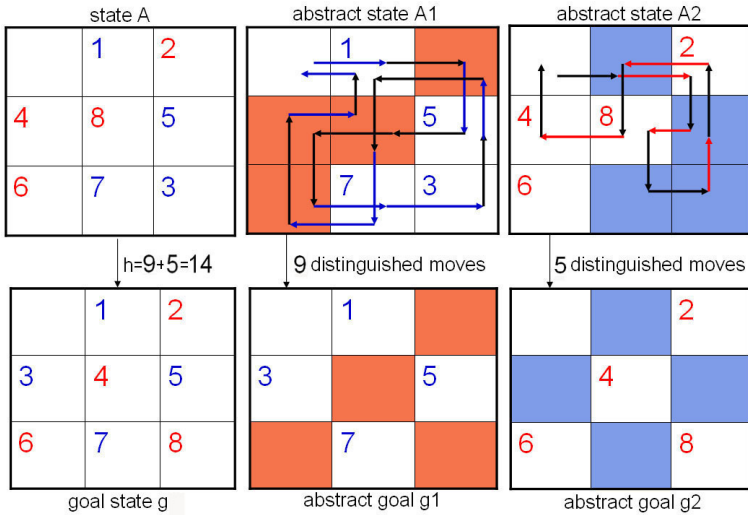


Fig. 1. The primary cost C^* is defined by the total moves of numbered tiles in the abstract state (i.e. distinguished moves) and the residual cost R^* is the number of moves of other tiles. $(C_1^*, R_1^*) = (9, 9), (C_2^*, R_2^*) = (5, 7)$. $h = \sum_{i=1}^2 C_i^* < (C_1^* + R_1^*)$. So $h=14$ is an infeasible heuristic value. h can be improved to be 16.

Table 1. 15 sliding tile puzzle results

Tile Partition	Check Infeasibility	Average H	Average Nodes	Average Sec
5-5-5	Yes	42.10	1,453,358	0.312
	No	41.56	3,186,654	0.642
6-6-3	Yes	42.78	784,145	0.171
	No	42.13	1,858,899	0.379

is over 2 times faster than the running time required on average without checking for infeasibility on the same machine.

References

- Edelkamp, S.: Planning with pattern databases. In: Proceedings of the 6th European Conference on Planning, pp. 13–34 (2001)
- Felner, A., Korf, E., Hanan, S.: Additive pattern database heuristics. *Journal of Artificial Intelligence Research*. 22, 279–318 (2004)
- Korf, E., Felner, A.: Disjoint pattern database heuristics. *Artificial Intelligence*. 134, 9–22 (2002)
- Prieditis, A.E.: Machine discovery of effective admissible heuristics. *Machine Learning* 12, 117–141 (1993)
- Yang, F., Culberson, J., Holte, R.: A general additive search abstraction. Technical Report TR07-06. Department of Computing Science, University of Alberta (2007)

Leveraging Graph Locality Via Abstraction

Rong Zhou

Intelligent Systems Laboratory
Palo Alto Research Center, Palo Alto CA 94304, USA
rzhou@parc.com

The use of abstraction to speedup problem solving is ubiquitous in AI, especially in the field of heuristic search where abstraction has proven a crucial technique for creating highly accurate memory-based heuristics known as pattern databases (PDBs). While PDBs are intrinsically based on problem abstractions [1], the converse is not necessarily true, and this suggests that abstraction should play a much bigger role than simply improving the quality of the heuristic.

This has inspired the development of a technique called *structured duplicate detection*, which uses abstraction to reveal as well as leverage the local structure of a search problem. Unlike PDBs, structured duplicate detection considers the neighborhood of an abstract state in the final search, and uses this information to localize memory references in duplicate detection. Using a locality-preserving abstraction function, structure duplicate detection can (i) limit the number of slow disk I/O operations in external-memory graph search, and (ii) reduce the synchronization (or communication) overhead in parallel graph search. The success of structured duplicate detection in areas such as disk-based search [2], external-memory heuristics [3], domain-independent STRIPS planning [4], and parallel graph search [5] speaks for the generality and effectiveness of using abstraction beyond its application to regular pattern databases.

An important focus of my research is on the exploitation of graph locality to improve the memory efficiency of various shortest-path algorithms, including breadth-first search, Dijkstra's algorithm, and A*. For example, breadth-first heuristic search [6] exploits the locality *between* different layers of a graph, in order to determine the minimum number of layers that must be stored in memory to ensure duplicate detection. Thus, algorithms such as breadth-first iterative-deepening A* [7] and divide-and-conquer beam-stack search [8] exploit a kind of *inter-layer* graph locality. On the other hand, because structured duplicate detection uses state-space abstraction to determine a subset of nodes *within* a layer that must be stored for duplicate detection, it exploits a kind of *inner-layer* graph locality.

A key advantage of using abstraction to exploit graph locality is that it offers a flexible way to uncover the kind (and amount) of local structure needed by structured duplicate detection. For example, changing the granularity of the abstraction function can effectively control the size of the largest duplicate-detection scope, which reflects the minimum internal-memory requirement of structured duplicate detection. The same approach can also be used to control the maximum number of independent processors that are allowed to perform concurrent duplicate detection without excessive synchronization in parallel graph search [5].

Another advantage of exploiting abstraction-based graph locality is that it is effective for both informed and uninformed search. This can prove particularly useful in areas such as domain-independent planning and model checking where the best admissible heuristic function is rather weak or expensive to compute.

Because good locality-preserving abstractions can be found automatically, this approach to leveraging local problem structure can be integrated with a domain-independent planner [4]. Although in the worst case a problem may not have any intrinsic local structure, using a technique called *edge partitioning* [9] it is always possible to create “artificial” graph locality that can be leveraged by structured duplicate detection, thereby making this approach fully general.

References

1. Zhou, R., Hansen, E.: Space-efficient memory-based heuristics. In: Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-04), pp. 677–682 (2004)
2. Zhou, R., Hansen, E.: Structured duplicate detection in external-memory graph search. In: Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-04), pp. 683–688 (2004)
3. Zhou, R., Hansen, E.: External-memory pattern databases using structured duplicate detection. In: Proc. of the 20th National Conference on Artificial Intelligence (AAAI-05) pp. 1398–1405 (2005)
4. Zhou, R., Hansen, E.: Domain-independent structured duplicate detection. In: Proc. of the 21st National Conference on Artificial Intelligence (AAAI-06), pp. 1082–1087 (2006)
5. Zhou, R., Hansen, E.: Parallel structured duplicate detection. In: Proc. of the 22nd National Conference on Artificial Intelligence (AAAI-07) (to appear)
6. Zhou, R., Hansen, E.: Breadth-first heuristic search. *Artificial Intelligence* 170, 385–408 (2006)
7. Zhou, R., Hansen, E.: Breadth-first heuristic search. In: Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-04), pp. 92–100 (2004)
8. Zhou, R., Hansen, E.: Beam-stack search: Integrating backtracking with beam search. In: Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05), pp. 90–98 (2005)
9. Zhou, R., Hansen, E.: Edge partitioning in external-memory graph search. In: Proc. of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07), pp. 2410–2416 (2007)

Author Index

- Abbott, Russ 391
Abril, Montserrat 5
Anderson, Kenneth 20
Anderson, Scot 35
- B, Ravindran 300
Barber, Federico 5
Barták, Roman 50
Barto, Andrew 273
Bayer, Kenneth M. 64
Beck, J. Christopher 393
Bessiere, Christian 80
Bulitko, Vadim 1
- Cambazard, Hadrien 395
Castro, Carlos 397
Choueiry, Berthe Y. 64, 402
Condotta, Jean-François 93
Crawford, Broderick 397
Culberson, Joseph 413
- D’Almeida, Dominique 93
de Kleer, Johan 109
De Saeger, Stijn 124
- Eyal, Amir 169
- Feldman, Alexander 139
Felner, Ariel 155
Fox, Maria 200
Frisch, Alan M. 2
- Gammer, Igor 169
Genesereth, Michael R. 215
Gent, Ian P. 184
Gregory, Peter 200
Gruninger, Michael 393
- Haley, David C. 399
Hebrard, Emmanuel 80
Hinrichs, Timothy L. 215, 400
Hnich, Brahim 80
Holte, Robert 20, 413
Hooker, John N. 4
Hu, Jiaqiao 243
- Ibrahim, Zina M. 230
- Jansen, Renee 344
Jonathan, Schaeffer 20
Jong, Nicholas K. 258
Jonsson, Anders 273
- Kiziltan, Zeynep 80
Knoblock, Craig A. 64, 402
Kuter, Ugur 243
- Lecoutre, Christophe 93
Long, Derek 200
- Macho González, Santiago 285
Meseguer, Pedro 285
Michalowski, Martin 64, 402
Miguel, Ian 184
Monfroy, Eric 397
- O’Sullivan, Barry 395, 411
Ofek, Nir 155
- Pickett, Marc 405
Provan, Gregory 139
- Quimper, Claude-Guy 80
- Raghavan, Sriram 300
Rendl, Andrea 184
Revesz, Peter 35
- Saïs, Lakhdar 93
Saitta, Lorenza 314, 375
Salido, Miguel A. 5, 407
Schachte, Peter 329
Schkufza, Eric 409
Shimajima, Atsushi 124
Søndergaard, Harald 329
Stone, Peter 258
Sturtevant, Nathan 344
Surynek, Pavel 359
- Tawfik, Ahmed Y. 230
Torasso, Pietro 314
Torta, Gianluca 314

van Gemund, Arjan 139

Vu, Xuan-Ha 411

Walsh, Toby 80

Yang, Fan 413

Zhou, Rong 415

Zucker, Jean-Daniel 375